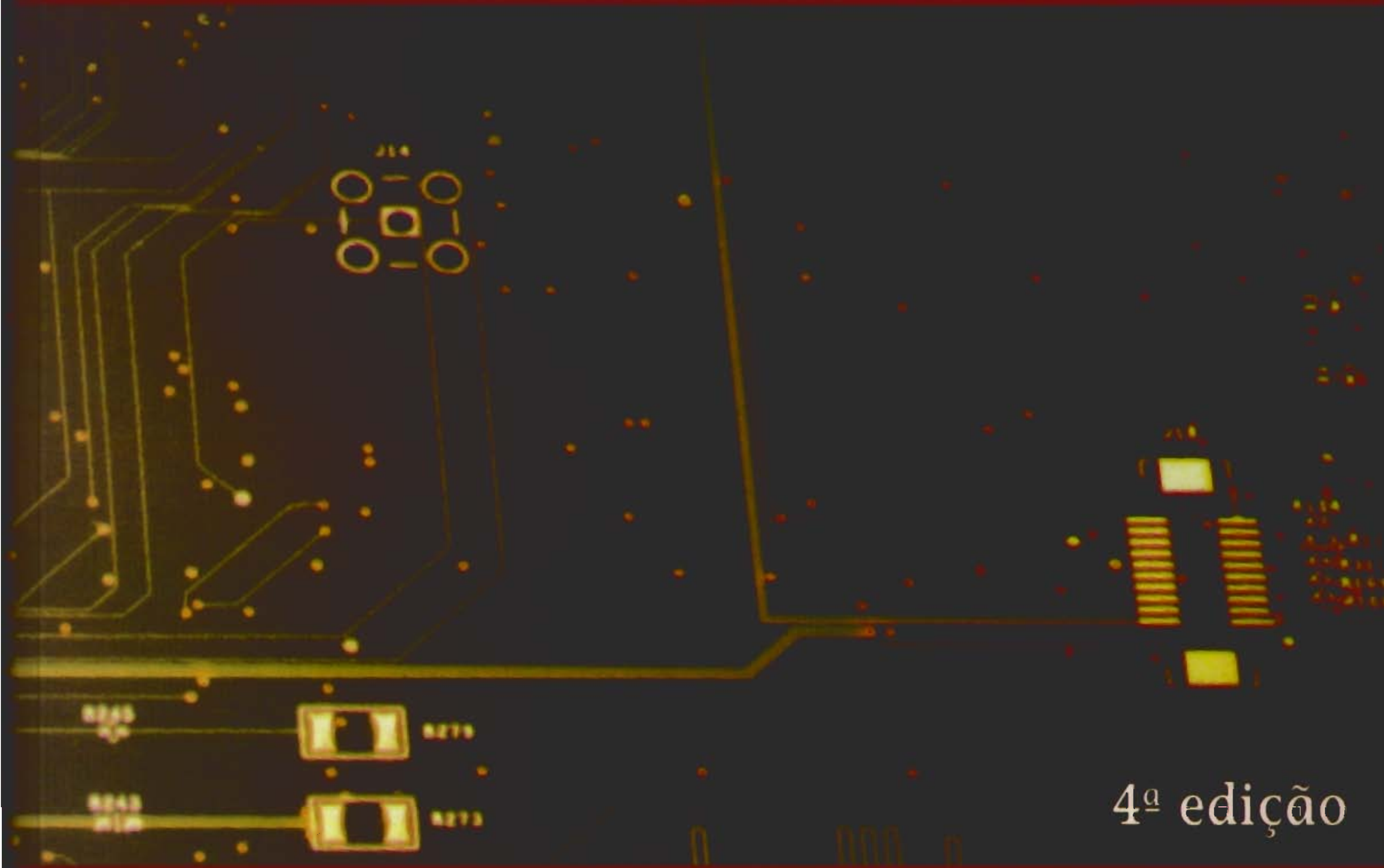
INTRODUÇÃO À ORGANIZAÇÃO DE COMPUTADORES



4ª edição

Mário A. Monteiro

UNIBAN
Tombo 00229379

LTC

INTRODUÇÃO À ORGANIZAÇÃO DE COMPUTADORES

ABPDEA
Associação Brasileira para
a Proteção dos Direitos
Editoriais e Autorais
RESPEITE O AUTOR
NAO FAÇA CÓPIA
www.abpdea.org.br

INTRODUÇÃO À ORGANIZAÇÃO DE COMPUTADORES

Mário A. Monteiro

Formado pela Escola Naval, com especialização em Eletrônica,
foi responsável pela área de suporte de sistemas da Marinha.
Mestre em Informática pela PUC-Rio, leciona sobre o assunto há mais de 21 anos.
Atualmente ministra cursos na Universidade Estácio de Sá.
Diretor de empresa de Informática e consultor de diversas outras empresas da área.

4.ª edição

UNIVERSIDADE BANDEIRANTE DE SÃO PAULO
BIBLIOTECA

LTC
EDITORA

Prefácio

Este livro, publicado pela primeira vez há cerca de oito anos, tem o propósito de servir de livro-texto básico para diversos tipos de cursos na área de Informática, seja no contexto de graduação, seja no de cursos de extensão universitária. O material nele contido e a forma de apresentação e descrição permitem, ainda, que ele possa ser utilizado até mesmo para estudos individuais de principiantes.

A sua receptividade no mercado e nos meios escolares motivou seu aperfeiçoamento, o que redundou nas edições subseqüentes.

A estrutura dos assuntos em capítulos procurou seguir a natural organização de um computador, com seus componentes básicos: memória, processador e entrada/saída. Para melhor entendimento, acrescentaram-se alguns capítulos com extensões intrínsecas e pertinentes aos citados componentes.

O conteúdo de cada capítulo foi elaborado e tem sido atualizado, considerando o programa e a experiência de ministrá-lo por mais de 20 anos em cursos na área de Informática em Universidades, empresas e outras instituições.

Para que o livro pudesse ser convenientemente adotado em uma multiplicidade de cursos com objetivos e profundidades diferentes, a maioria dos capítulos se divide em duas partes: a primeira, mais descritiva e básica, apresenta ao leitor as características e particularidades essenciais ao conhecimento principal do assunto título do capítulo, como o processador ou a memória. Uma segunda parte foi acrescentada, denominada "Um pouco mais de detalhe", em que se inserem tópicos adicionais ou uma descrição mais detalhada do assunto básico. Essa parte é destinada àqueles leitores mais curiosos ou que necessitam de informações mais profundas sobre um item específico.

O livro contém 11 capítulos e três Apêndices, além de um Glossário e a Bibliografia. Nesta última edição acrescentaram-se, também, as respostas aos exercícios apresentados em cada capítulo.

O Cap. 1 consiste em uma introdução ao assunto abordado, estabelecendo alguns conceitos básicos e concluindo com um breve histórico sobre a criação e evolução dos computadores. Nesta edição, o texto relativo ao histórico dos computadores foi ampliado.

O Cap. 2 apresenta de modo sucinto os componentes básicos de um computador e define unidades essenciais ao entendimento do resto do livro, como o bit e o byte.

Como os computadores digitais são máquinas binárias, e estamos habituados à aritmética decimal, no Cap. 3 são apresentados conceitos sobre sistemas de numeração não-decimais, bem como métodos para conversão de valores da base 10 para a base 2 e para as bases 8 e 16, usadas complementarmente nos sistemas de PD. Mostra-se, ainda, como se efetuam operações aritméticas em sistemas não-decimais. O capítulo sofreu uma revisão, de modo a tornar o texto mais claro e simples, aumentando-se também a quantidade de exercícios. Ainda assim, se o leitor estiver interessado em maiores detalhes, poderá encontrá-los no Apêndice A — Sistemas de Numeração, onde o mesmo assunto é tratado, porém com mais profundidade.

O Cap. 4, inserido posteriormente por sugestão de vários professores, contém uma descrição dos principais conceitos de Lógica Digital, alicerce no projeto e implementação de computadores digitais.

Os Caps. 5 e 6, que foram revistos e ampliados para esta edição, mostram, respectivamente, detalhes conceituais de memória e dos processadores, bem como de seu funcionamento integrado, através da execução de programas criados em linguagem de máquina. A revisão de ambos os capítulos visou à atualização das informações em face da crescente e contínua evolução da tecnologia de fabricação de memórias e processadores.

Os Caps. 7 e 8 abordam aspectos relativos ao modo como os dados e as instruções são representados internamente em um sistema de computação, bem como

Sumário

1 INTRODUÇÃO, 1

- 1.1 Conceituação, 1
    - 1.1.1 Processamento de Dados, 1
    - 1.1.2 Sistemas, 3
    - 1.1.3 Sistemas de Computação, 4
- 1.2 Histórico, 8
    - 1.2.1 Época dos Dispositivos Mecânicos (500 a.C. — 1880), 9
    - 1.2.2 Época dos Dispositivos Eletromecânicos (1880-1930), 11
    - 1.2.3 Época dos Componentes Eletrônicos — Primeiras Invenções (1930-1945), 12
    - 1.2.4 A Evolução dos Computadores Eletrônicos (1945 — até quando?), 13
        - 1.2.4.1 Primeira Geração — Computadores a Válvula, 13
        - 1.2.4.2 Segunda Geração — Computadores Transistorizados, 16
        - 1.2.4.3 Terceira Geração — Computadores com Circuitos Integrados, 17
        - 1.2.4.4 Quarta Geração — Computadores que Utilizam VLSI, 18
        - 1.2.4.5 Evolução dos Computadores de Grande Porte (Mainframes), 18
        - 1.2.4.6 Computadores Pessoais — Microcomputadores, 20
- Exercícios, 23

2 COMPONENTES DE UM SISTEMA DE COMPUTAÇÃO, 24

- 2.1 Descrição dos Componentes, 24
- 2.2 Representação das Informações, 28
    - 2.2.1 O Bit, o Caractere, o Byte e a Palavra, 28
    - 2.2.2 Conceito de Arquivos e Registros, 31
- 2.3 Classificação de Sistemas de Computação, 32
- 2.4 Medidas de Desempenho de Sistemas de Computação, 36
- Exercícios, 37

3 CONVERSÃO DE BASES E ARITMÉTICA COMPUTACIONAL, 39

- 3.1 Notação Posicional — Base Decimal, 39
- 3.2 Outras Bases de Numeração, 40
- 3.3 Conversão de Bases, 43
    - 3.3.1 Conversão entre Bases Potência de 2, 43
        - 3.3.1.1 Entre as Bases 2 e 8, 43
        - 3.3.1.2 Entre as Bases 2 e 16, 43
        - 3.3.1.3 Entre as Bases 8 e 16, 44
    - 3.3.2 Conversão de Números de uma Base B para a Base 10, 44
    - 3.3.3 Conversão de Números Decimais para uma Base B, 46
- 3.4 Aritmética Não-Decimal, 48
    - 3.4.1 Aritmética Binária, 49

        3.4.1.1 Soma Binária, 49
        3.4.1.2 Subtração Binária, 49
        3.4.1.3 Multiplicação Binária, 51
        3.4.1.4 Divisão Binária, 52
    3.4.2 Aritmética Octal (em Base 8), 55
    3.4.3 Aritmética Hexadecimal (em Base 16), 57
Exercícios, 58

4 CONCEITOS DA LÓGICA DIGITAL, 64
4.1 Introdução, 64
4.2 Portas e Operações Lógicas, 65
    4.2.1 Operação Lógica ou Porta AND (E), 67
    4.2.2 Operação Lógica ou Porta OR (OU), 69
    4.2.3 Operação Lógica NOT (Inversor), 71
    4.2.4 Operação Lógica NAND — NOT AND, 72
    4.2.5 Operação Lógica NOR — NOT OR, 75
    4.2.6 Operação Lógica XOR — EXCLUSIVE OR, 78
4.3 Expressões Lógicas — Aplicações de Portas, 80
    4.3.1 Cálculos com Expressões Lógicas, 81
4.4 Um Pouco Mais de Detalhe, 87
    4.4.1 Introdução, 87
    4.4.2 Noções de Álgebra Booleana, 87
    4.4.3 Circuitos Combinatórios, 93
        4.4.3.1 Exemplo Prático — Projeto de um Multiplicador de 2 Bits, 94
        4.4.3.2 Portas Wired-Or e Wired-And, 96
    4.4.4 Circuitos Integrados, 97
        4.4.4.1 Decodificador, 100
        4.4.4.2 Flip-Flops, 102
Exercícios, 105

5 SUBSISTEMAS DE MEMÓRIA, 108
5.1 Introdução, 108
    5.1.1 Como as Informações São Representadas nas Memórias, 110
    5.1.2 Como se Localiza uma Informação nas Memórias, 111
    5.1.3 Operações Realizadas em uma Memória, 111
5.2 Hierarquia de Memória, 113
    5.2.1 Registradores, 115
    5.2.2 Memória Cache, 117
    5.2.3 Memória Principal, 118
    5.2.4 Memória Secundária, 120
5.3 Memória Principal — MP, 122
    5.3.1 Organização da Memória Principal, 122
    5.3.2 Considerações sobre a Organização da Memória Principal, 124
    5.3.3 Operações com a Memória Principal, 125
        5.3.3.1 Operação de Leitura (Fig. 5.12), 127
        5.3.3.2 Operação de Escrita (Fig. 5.13), 128
    5.3.4 Capacidade de MP — Cálculos, 129
        5.3.4.1 Cálculos com Capacidade da MP (RAM), 131
    5.3.5 Tipos e Nomenclatura de MP, 135
        5.3.5.1 Memórias do Tipo ROM, 138
5.4 Erros, 141
5.5 Memória Cache, 142
    5.5.1 Conceitos, 143

                5.5.1.1 Diferença de Velocidade UCP/MP, 143
                5.5.1.2 Conceito de Localidade, 143
        5.5.2 Utilização da Memória Cache, 144
        5.5.3 Tipos de Memória Cache, 145
                5.5.3.1 Níveis de Cache de Memória RAM, 146
        5.5.4 Elementos de Projeto de uma Memória Cache, 147
                5.5.4.1 Tamanho da Memória Cache, 147
                5.5.4.2 Mapeamento de Dados MP/Cache, 148
                5.5.4.3 Algoritmos de Substituição de Dados na Cache, 153
                5.5.4.4 Política de Escrita pela Memória Cache, 154
    5.6 Um Pouco Mais de Detalhes, 155
        5.6.1 Sobre Tecnologias de Fabricação, 155
                5.6.1.1 Evolução da Tecnologia de Fabricação de Memórias DRAM, 157
                5.6.1.2 Sobre a Apresentação dos Elementos de Memória no Sistema de Computação, 159
        5.6.2 Localização da Célula Desejada em uma Operação de Leitura ou de Escrita, 160
                5.6.2.1 Organização de Memória do Tipo Seleção Linear — 1 Dimensão, 160
                5.6.2.2 Organização de Memória do Tipo Matriz Linha/Coluna, 164
    Exercícios, 165

6 UNIDADE CENTRAL DE PROCESSAMENTO, 168
    6.1 Introdução, 168
    6.2 Funções Básicas da UCP, 168
        6.2.1 Função Processamento, 172
                6.2.1.1 Unidade Aritmética e Lógica — UAL, 174
                6.2.1.2 Registradores, 175
                6.2.1.3 A Influência do Tamanho da Palavra, 177
        6.2.2 Função Controle, 181
                6.2.2.1 A Unidade de Controle, 182
                6.2.2.2 O Relógio, 184
                6.2.2.3 Registrador de Instrução (RI) — Instruction Register (IR), 186
                6.2.2.4 Contador de Instrução (CI) — Program Counter (PC), 186
                6.2.2.5 Decodificador de Instrução, 186
                6.2.2.6 Registrador de Dados de Memória — RDM e Registrador de Endereços de Memória — REM, 187
    6.3 Instruções de Máquina, 187
        6.3.1 O que É uma Instrução de Máquina, 187
        6.3.2 Formato das Instruções, 189
        6.3.3 Considerações sobre o Formato das Instruções, 190
    6.4 Funcionamento da UCP. O Ciclo da Instrução, 191
    6.5 Linguagem de Montagem (Assembly), 199
    6.6 Um Pouco Mais de Detalhe, 202
        6.6.1 Unidade Aritmética e Lógica — UAL, 203
        6.6.2 Metodologia Tipo Linha de Montagem ou *Pipelining*, 208
        6.6.3 Barramento, 212
                6.6.3.1 Comparação entre o Barramento Síncrono e Assíncrono, 219
        6.6.4 Tipos de Controle em um Processador, 219
                6.6.4.1 Controle Programado no Hardware, 220
                6.6.4.2 Controle por Microprogramação, 222
        6.6.5 Os Processadores e Suas Arquiteturas, 226
                6.6.5.1 A Evolução da Arquitetura X86 da Intel, 226
                6.6.5.2 Os Processadores de Outros Fabricantes, 241
        6.6.6 Elementos Auxiliares, 246

- 6.6.6.1 Circuitos de Apoio (Chipsets), 246
- 6.6.6.2 Encapsulamento dos Circuitos em uma Pastilha (Chip) e Soquetes para Inserção de Processadores nas Placas-mãe, 248
- 6.6.6.3 Overclocking, 251
- 6.6.6.4 Organização de Dados na Memória do Tipo Big Endian e Little Endian, 254
- Exercícios, 255

7 REPRESENTAÇÃO DE DADOS, 258

- 7.1 Introdução, 258
- 7.2 Tipos de Dados, 261
- 7.3 Tipo Caractere, 262
- 7.4 Tipo Lógico, 264
  - 7.4.1 Operador Lógico AND, 264
  - 7.4.2 Operador Lógico OR, 266
  - 7.4.3 Operador Lógico NOT, 268
  - 7.4.4 Operador Lógico EXCLUSIVE-OR (OU EXCLUSIVO), 269
- 7.5 Tipo Numérico, 272
  - 7.5.1 Representação em Ponto Fixo, 274
    - 7.5.1.1 Sinal e Magnitude, 275
    - 7.5.1.2 Representação de Números Negativos em Complemento, 282
  - 7.5.2 Overflow, 296
  - 7.5.3 Representação em Ponto Flutuante, 297
    - 7.5.3.1 Representação Normalizada, 299
    - 7.5.3.2 Conversão de Números para Ponto Flutuante, 300
  - 7.5.4 Representação Decimal, 307
- 7.6 Um Pouco Mais de Detalhe, 314
  - 7.6.1 Sobre a Representação em Ponto Flutuante, 314
  - 7.6.2 O Padrão IEEE-754, 1985, 315
- Exercícios, 318

8 REPRESENTAÇÃO DE INSTRUÇÕES, 322

- 8.1 Quantidade de Operandos, 324
  - 8.1.1 Instruções com Três Operandos, 326
  - 8.1.2 Instruções com Dois Operandos, 327
  - 8.1.3 Instruções com Um Operando, 329
- 8.2 Modos de Endereçamento, 330
  - 8.2.1 Modo Imediato, 331
  - 8.2.2 Modo Direto, 332
  - 8.2.3 Modo Indireto, 333
  - 8.2.4 Endereçamento por Registrador, 335
  - 8.2.5 Modo Indexado, 338
  - 8.2.6 Modo Base Mais Deslocamento, 343
- Exercícios, 345

9 EXECUÇÃO DE PROGRAMAS, 347

- 9.1 Introdução, 347
- 9.2 Linguagens de Programação, 347
- 9.3 Montagem e Compilação, 350
  - 9.3.1 Montagem, 351
  - 9.3.2 Compilação, 353
- 9.4 Ligação ou Linkedição, 357
- 9.5 Interpretação, 359

        9.5.1 Compilação × Interpretação, 360
    9.6 Execução de Programas em Código de Máquina, 362
    Exercícios, 369

10 ENTRADA E SAÍDA (E/S), 371
    10.1 Introdução, 371
    10.2 Interfaces de E/S, 377
        10.2.1 Transmissão Serial, 379
            10.2.1.1 Transmissão Assíncrona, 380
            10.2.1.2 Transmissão Síncrona, 382
        10.2.2 Transmissão Paralela, 383
    10.3 Dispositivos de E/S, 384
        10.3.1 Teclado, 384
        10.3.2 Monitor de Vídeo, 386
            10.3.2.1 Modalidade Textual ou Símbolo a Símbolo, 390
            10.3.2.2 Modalidade Gráfica ou Bit a Bit, 392
            10.3.2.3 Vídeo Colorido, 393
            10.3.2.4 Algumas Observações Finais, 393
        10.3.3 Impressoras, 395
            10.3.3.1 Impressoras Matriciais, 395
            10.3.3.2 Impressoras de Jato de Tinta, 397
            10.3.3.3 Impressoras a Laser, 397
        10.3.4 Fitas Magnéticas, 399
        10.3.5 Discos Magnéticos, 402
            10.3.5.1 Discos Flexíveis ou Disquetes (Floppy-Disk), 405
            10.3.5.2 Unidades de Disco do Tipo Winchester, 405
            10.3.5.3 Cálculo de Espaço de Armazenamento em Discos, 406
        10.3.6 Mouse, 407
    10.4 Métodos de Realização de Operações de E/S, 408
        10.4.1 Entrada/Saída por Programa, 409
        10.4.2 Entrada e Saída com Emprego de Interrupção, 411
        10.4.3 Acesso Direto à Memória — DMA, 413
    10.5 Um Pouco Mais de Detalhe, 414
        10.5.1 Introdução, 414
        10.5.2 Teclado, 415
            10.5.2.1 Etapas Básicas do Funcionamento de um Teclado Utilizado em Microcomputadores, 415
            10.5.2.2 Um Pouco de História, 417
        10.5.3 Outras Informações sobre Vídeos, 418
            10.5.3.1 Vídeos de Cristal Líquido — LCD, 419
            10.5.3.2 Vídeos de Gás Plasma, 421
        10.5.4 Tecnologias Alternativas para Impressão em Cores, 421
        10.5.5 Scanners, 423
        10.5.6 Dispositivos de Armazenamento Ótico, 424
    Exercícios, 425

11 ARQUITETURAS RISC, 428
    11.1 Introdução, 428
    11.2 Características das Arquiteturas RISC, 430
        11.2.1 Menor Quantidade de Instruções e Tamanho Fixo, 430
        11.2.2 Execução Otimizada de Chamada de Funções, 431
        11.2.3 Menor Quantidade de Modos de Endereçamento, 431
        11.2.4 Modo de Execução com *Pipelining*, 431

11.3 Medidas de Desempenho, 432
11.4 Observações a Respeito de CISC × RISC, 433
11.5 Sistemas RISC Comerciais, 434
    11.5.1 Processadores SPARC, 435
    11.5.2 Processadores Power RS/6000, 436
    11.5.3 Processadores ALPHA, 438
Exercícios, 438

APÊNDICES, 439
A - SISTEMAS DE NUMERAÇÃO, 439
A.1 Sobre Símbolos e Números, 439
A.2 Sistema de Numeração Não-Posicional, 440
A.3 Sistema de Numeração Posicional, 441
    A.3.1 Base, 442
    A.3.2 Um Pouco de História, 443
A.4 Algarismos e Números, 444
A.5 Conversão de Bases, 445
    A.5.1 Da Base 10 para uma Base B Qualquer, 445
        A.5.1.1 Conversão de Números Inteiros, 445
        A.5.1.2 Conversão de Números Fracionários, 447
    A.5.2 Conversão de Base B (não 10) para Valor Decimal, 449
    A.5.3 Conversão Direta entre Bases Não-decimais, 450
A.6 Outros Métodos de Conversão de Bases, 451
A.7 Operações Aritméticas, 454
    A.7.1 Procedimentos de Adição, 454
        A.7.1.1 Adição de Números Binários, 454
        A.7.1.2 Adição de Números Octais e Hexadecimais, 455
    A.7.2 Procedimentos de Subtração, 456
        A.7.2.1 Subtração de Números Binários, 456
        A.7.2.2 Subtração de Números Octais e Hexadecimais, 457
    A.7.3 Multiplicação de Números Binários, 457
    A.7.4 Divisão de Números Binários, 458

B - CÓDIGOS DE REPRESENTAÇÃO DE CARACTERES, 460

C - GLOSSÁRIO, 463

BIBLIOGRAFIA, 470

RESPOSTAS DOS EXERCÍCIOS, 472

ÍNDICE, 496

1

Introdução

1.1 CONCEITUAÇÃO

1.1.1 Processamento de Dados

Um computador é uma máquina (conjunto de partes eletrônicas e eletromecânicas) capaz de sistematicamente coletar, manipular e fornecer os resultados da manipulação de informações para um ou mais objetivos. Por ser uma máquina composta de vários circuitos e componentes eletrônicos, o computador também é chamado de equipamento de *processamento eletrônico de dados.*

Processamento de dados (*Data Processing*) consiste, então, em uma série de atividades ordenadamente realizadas, com o objetivo de produzir um arranjo determinado de informações a partir de outras obtidas inicialmente.

A manipulação das informações coletadas no início da atividade chama-se *processamento*; as informações iniciais são usualmente denominadas *dados*.

Os termos *dado* e *informação* podem ser tratados como sinônimos ou como termos distintos; *dado* pode ser definido como a matéria-prima originalmente obtida de uma ou mais fontes (etapa de coleta), e *informação* como o resultado do processamento, isto é, o dado processado ou "acabado".

A Fig. 1.1 mostra o esquema básico de um processamento de dados (manual ou automático), que resulta em um produto acabado: *a informação.*

Informação subentende dados organizados (segundo uma orientação específica) para o atendimento ou emprego de uma pessoa ou grupo que os recebe.

Como o conhecimento e a tomada de decisão são importantes em várias áreas e em diferentes níveis hierárquicos de uma organização, a informação para uma determinada pessoa ou grupo pode ser considerada um dado para outra.

Por exemplo, o processamento eletrônico de dados (PED) de itens do estoque de uma empresa pode estar estruturado para ser realizado em diferentes etapas. Na primeira, deseja-se apenas atualizar as informações de estoque para uso do almoxarifado e, nesse caso, os *dados* (de entrada) são itens recebidos e retirados em um dia, bem como a posição do dia anterior; o *processamento* consistirá, basicamente, em operações aritméticas de soma

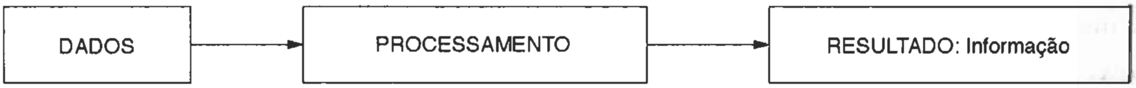


Figura 1.1 Etapas básicas de um processamento de dados.

e subtração (além de outras não-principais); como *resultado* (de saída), obtêm-se informações sobre a nova posição do estoque.

Numa segunda etapa, pode-se ter um outro tipo de processamento, neste caso, para produzir informações para um outro nível de tomada de decisão. Assim, utiliza-se como *dados* a posição do estoque (informação no processamento anterior); o *processamento* verificará quais itens estão abaixo de um mínimo e como *resultado* (de saída) obtém-se a nova informação (itens especificamente selecionados).

É claro que, nesse exemplo, um único processamento poderá obter as duas informações, mas isso não impede que constatemos a variação do emprego de dado e informação.

A obtenção de dados e a realização de seu processamento para produzir informações específicas são uma atividade que vem sendo exercida desde os primórdios da civilização. O que tem variado com o correr do tempo é, além naturalmente da tecnologia, o volume de dados a ser manipulado e a eficácia da manipulação, medida em termos de velocidade e flexibilidade na obtenção das informações resultantes.

A busca de técnicas mais eficazes de processamento de dados, aliada ao natural avanço tecnológico em diversos outros ramos de atividade, como a eletrônica e a mecânica, por exemplo, conduziu o mundo ao desenvolvimento de equipamentos de PED — os computadores — capazes de coletar, armazenar e processar dados muito mais rapidamente que os antigos meios manuais. Como veremos mais adiante neste capítulo, o primeiro computador (a máquina capaz de automatizar o processamento de dados) surgiu da necessidade de se acelerar a realização de cálculos (processamento matemático de dados).

Atualmente, com a imensa quantidade de informações que precisam ser conhecidas e atualizadas rapidamente pelas empresas, a utilidade dos computadores deixou de ser apenas importante para se tornar essencial, quase imprescindível, em praticamente todo tipo de atividade.

O avanço tecnológico na área de telecomunicações também contribuiu de modo considerável para o crescimento do uso de computadores, visto que permitiu sua interligação, criando-se as redes de comunicação de dados, tanto as internas a uma empresa quanto aquelas que interligam outras redes. O exemplo mais claro e de conhecimento mais amplo do público em geral é o da Internet, a rede mundial capaz de permitir a comunicação entre praticamente todos os tipos de computador, alavancando, também, seu conhecimento e sua utilização em áreas da sociedade muitas vezes distantes de qualquer tipo de tecnologia.

A organização de um computador é a parte do estudo da ciência da computação que trata dos aspectos relativos à parte do computador mais conhecida dos especialistas que o construíram e cujo entendimento é desnecessário ao programador. São aspectos de hardware específicos, como a tecnologia utilizada na construção da memória, a freqüência do relógio, os sinais de controle para iniciar as microoperações nas diversas unidades da máquina.

A arquitetura do mesmo computador é uma outra parte da mesma ciência da computação, mais no nível de conhecimento desejado pelo programador, visto que suas características (as da arquitetura de uma determinada máquina) têm impacto direto na elaboração de um programa. São elementos de uma arquitetura o conjunto de instruções de um processador, o tamanho da palavra, os modos de endereçamento das instruções, o tipo e o tamanho dos dados manipulados pelo processador.

Como exemplo da distinção entre essas partes, podemos mencionar como elemento de decisão de uma dada arquitetura se a sua unidade de controle será "programada por hardware" ou se será microprogramada (ver Cap. 6). Decidido, por exemplo, que ela será microprogramada, então um aspecto da organização do processador a ser decidido refere-se ao tipo de tecnologia e tamanho da memória de controle, que armazenará as microinstruções projetadas para o referido processador.

Desse modo, um fabricante pode definir elementos característicos da arquitetura de uma "família" de processadores e construir vários deles, cada um com uma diferente organização, com um modelo diferente para venda. Um exemplo de "família" de processadores com uma dada arquitetura é a do Intel x86.

Na nossa vida cotidiana, costumamos perceber essas diferenças entre arquitetura e organização em outros ramos de atividade, embora não se utilize a mesma nomenclatura aqui apresentada. É o caso, por exemplo, da especificação da arquitetura de um edifício (como a quantidade de cômodos, o tipo de cobertura das paredes da cozinha, se terá ou não varanda e outros). Dados semelhantes aos aspectos de organização já indicados são os relativos a estrutura, cálculos de peso e de vigas, encanamentos etc.

Ao longo do tempo, diversos tipos de arquitetura de computadores foram definidos, de modo que é possível criar-se classificações nas quais cada computador projetado pode se enquadrar. Uma das classificações mais conhecidas foi estabelecida em 1972 por Michael Flynn [FLYN 72]. A classificação, que será sumarizada a seguir, refere-se à maneira como as instruções e os dados por elas manipulados estão organizados no processador. Todos os quatro tipos definidos por Flynn mencionam "trem de instruções — instruction stream" e "trem de dados — data stream".

Um trem de instruções é definido como sendo um conjunto de instruções seqüencialmente organizadas e executadas por um único processador, e um trem de dados é definido como um fluxo de dados seqüencialmente organizado requerido pelo conjunto de instruções.

a) SISD (Single Instruction stream, Single Data stream) — Um único conjunto de instruções e de dados. Enquadram-se nessa classificação máquinas cuja arquitetura segue o padrão definido por von Neumann (ver item 1.2.4), ou seja, processadores que executam uma instrução completa de cada vez, seqüencialmente. Essas máquinas usam no processador um único registrador (CI — Contador de Instruções ou PC — Program Counter), como veremos no Cap. 6, que armazena o endereço da próxima instrução a ser executada e que vai automaticamente sendo incrementado para o próximo endereço de instrução. Atualmente não deve existir projeto ou fabricação de processadores SISD, devido à sua lentidão e ao atraso tecnológico.

b) MISD (Multiple Instruction stream, Single Data stream) — Este tipo de arquitetura estabelece que várias instruções podem ser executadas simultaneamente, manipulando um único conjunto de dados. Processadores vetoriais podem ser enquadrados nessa categoria de arquitetura.

c) SIMD (Single Instruction stream, Multiple Data stream) — Neste tipo de arquitetura, o processador opera de modo que uma única instrução acessa e manipula um conjunto de dados simultaneamente. Nesse caso, a unidade de controle do processador aciona diversas unidades de processamento. Como exemplo, pode-se imaginar o processamento 1000 vezes do seguinte comando: A (I) * B (I). Em um processador do tipo SISD este comando seria executado 1000 vezes, um após o outro, enquanto em um processador do tipo SIMD ele poderia ser executado em uma única vez, considerando que a máquina tivesse 1000 processadores e estes, então, realizariam um processamento paralelo dos 1000 As vezes os 1000 Bs.

d) MIMD (Multiple Instruction stream, Multiple Data stream) — É a categoria mais avançada tecnologicamente e que produz, se implementada, o melhor desempenho de processamento.

1.1.2 Sistemas

Um sistema pode ser definido de diferentes maneiras. Um sistema pode ser compreendido, por exemplo, como um conjunto de partes que cooperam para atingir-se um objetivo comum. Porém, a que parece mais apropriada para nossas conceituações é a seguinte, ligeiramente diferente da anterior:

"Conjunto de partes coordenadas que concorrem para a realização de um determinado objetivo."

Atualmente, o enfoque sistêmico se faz presente em quase todas as áreas do desenvolvimento comercial, científico, industrial e social.

Temos conhecimento do sistema de transporte de uma cidade, um conjunto de partes (os ônibus, as ruas, os motoristas e trocadores, as pessoas) que se integram (através das normas e trajetos aprovados pela Prefeitura da cidade) para atingir o objetivo de transportar pessoas de um local para outro de forma eficaz (o que nem sempre acontece...). Conhecemos outros sistemas, como o sistema circulatório do corpo humano, o sistema econômico do país (!!!!) etc.

O processamento eletrônico de dados, devido a sua própria natureza — a de ser um conjunto de componentes separados que se integram segundo procedimentos e regras previamente estabelecidos —, vem se desenvolvendo de acordo com os conceitos da Teoria de Sistemas e, por essa razão, é chamado de sistema de computação.

É sistema porque é um conjunto de partes que se coordenam (o teclado, a memória, o processador, os dispositivos periféricos) para a realização de um objetivo: computar (por isso é de computação).

Computar significa *calcular*, realizar cálculos matemáticos. Os computadores são máquinas de computar, de calcular, de realizar operações matemáticas. O primeiro computador, desenvolvido na década de 1940, tinha o objetivo de acelerar cálculos balísticos para o Exército americano (ver item 1.2). E daí em diante, os computadores não pararam de evoluir tecnologicamente, mas continuaram sendo equipamentos para computar. Mesmo quando um processador está sendo usado para processar texto, ele o faz por meio de cálculos matemáticos; como também acontece quando ele realiza processamento gráfico e outros mais.

Calcular, realizar operações matemáticas, é uma tarefa a ser executada com valores, com dados. O cálculo matemático com valores numéricos nada mais é do que a manipulação desses valores. O resultado de uma operação matemática é normalmente armazenado em uma célula de memória ou registrador do processador, o que significa outro tipo de manipulação dos dados. E assim por diante. Na realidade, um computador realiza contínuas e constantes manipulações de dados. Chama-se a isso processamento de dados. A manipulação dos dados, realizada segundo instruções de um programa, é conhecida como processamento dos dados.

Sistemas de processamento de dados são aqueles responsáveis pela coleta, armazenamento, processamento e recuperação, em equipamentos de processamento eletrônico, dos dados necessários ao funcionamento de um outro sistema maior: o sistema de informações.

O sistema de informações de uma empresa pode ser conceituado como o conjunto de métodos, processos e equipamentos necessários para se obter, processar e utilizar informações dentro da empresa. Dessa forma, ele compreende não só o SPD (Sistema de Processamento de Dados), como também todos os procedimentos manuais necessários a prover informações para um determinado nível de decisão de uma organização.

Em qualquer organização, os sistemas de informações se desenvolvem segundo duas dimensões: os *componentes da organização*, isto é, seus diversos setores funcionais, e o *nível de decisão*, o qual obedece a uma hierarquia clássica, de níveis:

- nível operacional (de execução corriqueira e imediata, de competência dos menores escalões);
- nível gerencial (de nível intermediário, de competência da gerência setorial);
- alto nível da organização (de nível estratégico, de competência da diretoria).

O tipo de decisão tomada em cada nível requer um diferente grau de agregação da informação e, em conseqüência, diferentes tipos de relatórios e/ou apresentação e uso da informação.

Dentro deste enfoque, um sistema de informações gerenciais (SIG) pode ser conceituado como o sistema de informação que engloba todos os componentes e todos os níveis de decisão de uma organização.

Em geral, um sistema de processamento de dados compreende duas partes: o sistema de computação (o computador e os programas básicos) e os sistemas de aplicação. O primeiro, normalmente fornecido completo pelo fabricante ou fornecedores específicos, e os últimos, desenvolvidos pelo usuário ou por terceiros, especificamente dedicados a uma aplicação de interesse do usuário.

1.1.3 Sistemas de Computação

Qualquer processamento de dados requer a execução de uma série de etapas, as quais podem ser realizadas de forma manual ou automática por um computador. Tais etapas, elaboradas e executadas passo a passo, constituem o que se chama *programa*. Cada um dos passos mencionados é uma diferente instrução, ou ordem de comando, dada ao hardware, objetivando a realização de uma determinada ação (uma operação aritmética, uma transferência de informação etc.). O programa é o conjunto de instruções.

Consideremos que se deseja, por exemplo, somar 100 números e imprimir o resultado. Se o processo é manual, precisa-se de uma máquina de somar e outra de escrever, bem como de uma pessoa que executará todas as etapas. Estas poderão estar relacionadas em um papel, de modo que o operador não cometa erros nem se esqueça de alguma etapa, devendo ser executadas sistematicamente, uma após outra, conforme mostrado na Fig. 1.2.

Tal programa não é, entretanto, possível de ser diretamente executado pela máquina, uma vez que as linguagens de programação são apenas um modo de o operador comunicar-se com o computador. A máquina somente entende e executa instruções mais simples, chamadas instruções de máquina.

Todo computador é construído com circuitos eletrônicos capazes de reconhecer e executar diretamente apenas um conjunto limitado e simples de instruções de máquina, nas quais todo programa (escrito em Pascal, C, Delphi, Java etc.) deve ser convertido antes de ser executado. Essas instruções são normalmente do tipo:

- executar operações aritméticas sobre dois números;
- executar operações lógicas sobre dois números;
- mover um conjunto de bits (um número ou parte) de um local para outro do computador;
- desviar a seqüência do programa;
- fazer a comunicação com algum dispositivo de entrada ou saída de dados.

Em resumo, o computador, sendo uma máquina, precisa de "ordens" específicas (suas instruções) para executar as atividades para as quais foi construído. Durante seu projeto, seu idealizador definiu que sua máquina deveria realizar um conjunto de operações básicas, como, por exemplo: "somar dois números", "multiplicar dois números", "mover um dado de uma área da memória para o processador", "transferir um dado da memória para um periférico" etc. Essas operações básicas (muitas vezes conhecidas como operações primitivas) são formalizadas como instruções da máquina, sendo sua definição formal convertida em um conjunto de bits e sua implementação estabelecida no processador, que já vem programado da fábrica (o método de implementar as instruções primitivas em um processador pode variar (microprogramas é um tipo, p. ex.) conforme veremos no Cap. 6.

Esta formalização em instruções de máquina constitui a "linguagem" de comunicação dos humanos com a máquina e desta internamente entre seus componentes.

Assim como os humanos possuem uma linguagem própria, criada e desenvolvida ao longo do tempo para permitir nossa comunicação cotidiana (português, inglês, hebraico, francês, espanhol, alemão são exemplos de linguagens dos humanos), também foi necessário criar uma linguagem, aliás existem diversas, para comunicação dos computadores, a primeira das quais e mais básica é denominada linguagem de máquina e é imprescindível para o funcionamento de qualquer computador.

No entanto, como já mencionamos anteriormente, criar programas (organizar centenas, milhares, de instruções binárias, cheias de 0s e 1s) se tornou praticamente inviável em termos de perda de tempo, custo de pessoal, entre outros problemas, razão por que surgiram as linguagens chamadas de "alto nível" ou "orientada a aplicação". O nome alto nível decorre naturalmente do fato de essas linguagens serem mais distantes da forma de entendimento do processador e mais próximas do entendimento do programador, do ser humano, tendo características semelhantes às nossas linguagens de comunicação. Dadas as semelhanças com as linguagens dos humanos, que permitem ao programador pensar usando formas às quais ele já está habituado, tais como palavras da sua língua (as linguagens mais conhecidas foram desenvolvidas utilizando-se palavras da língua inglesa), sinais de operações matemáticas (+, −, /, *), seu emprego tornou-se mais vantajoso do que a utilização de símbolos pouco inteligíveis.

A Fig. 1.4 mostra um trecho de programa criado em três linguagens diferentes, com a finalidade de mostrar ao leitor, neste estágio inicial de entendimento sobre um computador e suas características, as diferenças mais visíveis entre elas, sendo clareza e concisão as principais. Como se pode observar nos exemplos da figura, o trecho do programa escrito em linguagem de alto nível possui muito menos linhas de código do que o mesmo programa convertido para a linguagem binária da máquina.

Uma outra vantagem dessas linguagens reside no fato de que, em face de suas semelhanças com as linguagens dos humanos, elas podem ser definidas para atender a requisitos e intenções específicas de emprego. Assim é que a linguagem Cobol foi definida para emprego em programas comerciais, a Fortran é mais bem utilizada em programação científica, além das linguagens surgidas mais recentemente, com o desenvolvimento dos sistemas gráficos (Visual Basic, Visual C) e assim por diante.

Ainda há outras vantagens do uso de linguagens de alto nível sobre a linguagem de máquina, entre as quais:

a) a concisão das linguagens de alto nível se comparadas com programas binários (ver Fig. 1.4). Neste último caso, os programas são mais longos e de difícil entendimento em face de as instruções serem mais simples e, portanto, cada item operacional precisar de muitas. Por exemplo: em uma linguagem de alto nível teríamos o seguinte comando:

X:= A + B / (C * D − A);

Linguagem Delphi

```
Procedure Tform1.TesteAsm;
var I, Total:Integer;
begin
  Total:=0;
  For I:=1 To 5 do
      Total:=Total+10;
end;
```

Linguagem Assembly

```
 push ebp
 mov ebp, esp
 add esp, -$0c
 mov [ ebp-$04] , eax

 xor eax, eax
 mov [ ebp-$0c] , eax

 mov[ ebp-$08] , $00000001

 add dword ptr [ ebp-$0c] , $0a

 inc dword ptr [ ebp-$08]
 cmp dword ptr [ epb-$08] , $06
 jnz TForm1.TestAsm + $15

 mov esp, ebp
 pop ebp
 ret
```

Linguagem de Máquina (binário)

```
01010101
0001011111101100
10000011111000100111110100
10001001010001011111111100
0011001111001101
10001001010001011111111100
110001110100010111111000001000000
1000001101000101111101000000001010

11111111010001011111111000
1000001101111101111111000000000110
0111010111110011
1000101111100101
01011101
11000011
```

Figura 1.4 Exemplo de programas em Delphi, Assembly e linguagem de máquina (binário).

o qual poderia se transformar nas seguintes instruções binárias:

1000111000000111

0000111000001100

0111111000001111

0101111000001001

1100111000001110

0011111000011000

0111111000011011

1110111000100011

O exemplo é ilustrativo das diferenças apontadas e já observadas na Fig. 1.4.

b) As linguagens de alto nível possuem uma estrutura de comandos que são identificados por palavras da nossa linguagem, o que facilita o entendimento do programador (usar SE/ENTÃO/SENÃO é mais intuitivo do que 01011011011).

Para efeitos de entendimento, desenvolvimento, fabricação e funcionamento, pode-se separar um computador em duas grandes e distintas partes: o *hardware* e o *software.*

O conjunto formado pelos circuitos eletrônicos e partes eletromecânicas de um computador é conhecido como *hardware* (ainda não há uma tradução adequada para esta palavra). É a parte física, visível do computador.

Software consiste nos programas, de qualquer tipo e em qualquer linguagem, que são introduzidos na máquina para fazê-la trabalhar, passo a passo, e produzir algum resultado — o conjunto de instruções, já mencionadas anteriormente. O hardware sozinho não funciona sem as instruções (software) sobre o quê e quando fazer.

O software dá vida ao computador, fazendo com que suas partes elétricas, eletrônicas e mecânicas possam funcionar. Pode-se fazer uma analogia com um automóvel: o carro em si é o hardware, poderoso, mas inerte até que uma pessoa possa ligar a chave e executar um conjunto de passos que o farão movimentar-se e atingir os objetivos para o qual foi construído. Nesse caso, a pessoa exerce a função do software, dando vida ao carro.

Há uma classe de programas, geralmente escritos pelo fabricante do computador ou por firmas especializadas, chamada genericamente *software básico* (é o caso dos sistemas operacionais, como o MS-DOS, o Windows, o Unix, como também dos compiladores e interpretadores — ver Cap. 9) e que, em conjunto com o hardware, constitui o que chamamos de *sistema de computação,* assunto deste trabalho.

A outra classe de programas, definida e implementada para atender aos problemas de cada um, é genericamente conhecida como *programas de aplicação* ou sistemas de aplicação, pois aplicam-se a um determinado problema de uma empresa ou mesmo de uma pessoa física.

1.2 HISTÓRICO

Embora o conhecimento histórico da evolução dos computadores não seja essencial para compreender seu funcionamento, é interessante que o leitor possa ter oportunidade de acompanhar historicamente seu desenvolvimento.

É comum encontrar em livros sobre o assunto uma divisão histórica da evolução dos computadores segundo o elemento eletrônico básico de sua organização: válvulas, transistores, circuitos integrados, pastilhas de alta e muito alta integração (VLSI).

No entanto, a idéia do uso de métodos de computação não é tão nova quanto o aparecimento da eletricidade e eletrônica.

1.2.1 Época dos Dispositivos Mecânicos (500 a.C. — 1880)

O conceito de efetuar cálculos com algum tipo de equipamento data, pelo menos, do século V a.C. com os babilônios e sua invenção do *ábaco*.

Esse dispositivo (ver Fig. 1.5) permitia a contagem de valores, tornando possível aos comerciantes babilônicos registrar dados numéricos sobre suas colheitas. Também os romanos se serviram muito dos ábacos, para efetuar cálculos aritméticos simples, registrando valores de outra forma (ver Apêndice A para explicação sobre algarismos romanos). Ainda hoje há quem use tal tipo de dispositivo, ainda popular na China, por exemplo.

A primeira evolução do ábaco surgiu somente no século XVII (em 1642), quando o filósofo e matemático francês Blaise Pascal construiu um contador mecânico que realizava operações aritméticas de soma e subtração por meio de rodas e engrenagens dentadas. Este instrumento consistia em seis engrenagens dentadas, com um ponteiro indicando o valor decimal escolhido ou calculado. Cada engrenagem continha 10 dentes que, após efetuarem um giro completo, acarretavam o avanço de um dente de uma segunda engrenagem, exatamente o mesmo princípio de um odômetro de automóvel e base de todas as calculadoras mecânicas. Cada conjunto de ponteiros era usado como um *registrador* para armazenar temporariamente o valor de um número. Um registrador atuava como *acumulador,* para guardar resultados e uma parcela. O outro registrador era utilizado para se introduzir um valor a ser somado ou subtraído do valor armazenado no acumulador. Quando se acionava a manivela e a máquina era colocada em movimento, os dois valores eram adicionados e o resultado aparecia no acumulador. O calculador de Pascal apresentou duas significativas inovações tecnológicas para sua época:

1) permitia o uso de "vai 1", passado automaticamente para a parcela seguinte; e
2) utilizava o conceito de complemento para realizar operações aritméticas de subtração através de soma de complemento (este conceito é até hoje essencialmente a base de funcionamento dos circuitos de operação aritmética em ponto fixo dos computadores).

A máquina, embora rudimentar, era eficaz para sua época, sendo inteiramente mecânica e não automática (funcionava por comando de uma manivela acionada manualmente). A linguagem de programação PASCAL foi assim chamada em homenagem a este cientista pelo seu trabalho pioneiro em matemática e também devido à sua invenção.

Algum tempo após a invenção de Pascal, outro filósofo e matemático, desta vez alemão, Gottfried Leibniz, construiu uma calculadora mais completa que a de Pascal, porque realizava as quatro operações aritméticas, e não apenas adição e subtração, como na de Pascal. O calculador mecânico de Leibniz era uma duplicata do calculador de Pascal acrescido de dois conjuntos de rodas, as quais permitiam a realização de multiplicações e divisões por meio de um processo de operações sucessivas (sabemos ser possível realizar multiplicações por somas sucessivas e divisão por subtrações sucessivas). Ambas as máquinas, a de Pascal e a de Leibniz, eram manuais.

A primeira utilização prática de dispositivos mecânicos para computar dados automaticamente data do início de 1800. Por esta época, muita contribuição para o desenvolvimento inicial dos computadores foi obtida do aperfeiçoamento de processos em uma área inteiramente diferente de computação — a de tecelagem. Em 1801, Joseph Jacquard produziu com sucesso um retrato em tecelagem, cuja produção foi inteiramente realizada de forma mecânica e controlada automaticamente por instruções registradas em orifícios em cartões perfu-

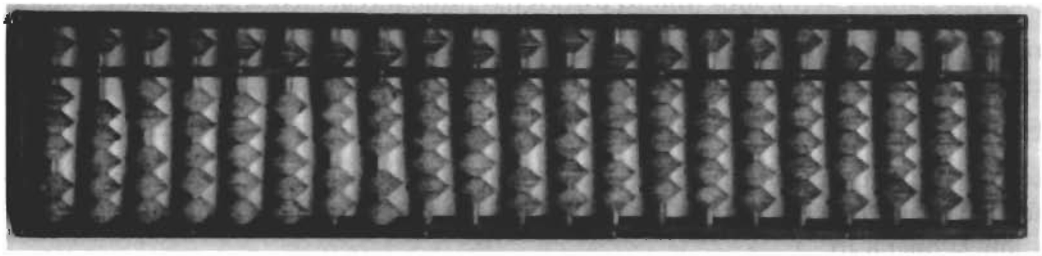


Figura 1.5 Um ábaco.

rados. Um dos mais conhecidos resultados deste processo, de realização de tarefas de tecelagem por uma máquina controlada por programa armazenado, foi o retrato do próprio Joseph Jacquard em uma tapeçaria, o qual consumiu 24.000 cartões.

Um dos últimos e mais importantes trabalhos pioneiros em computação por processos mecânicos foi realizado por um inglês de nome Charles Babbage, que, em 1823, foi contratado pela Royal Astronomical Society of Great Britain para produzir uma máquina calculadora programável, com a finalidade de gerar tabelas de navegação para a Marinha britânica.

Em seu trabalho, Babbage projetou dois tipos de máquinas: a *máquina de diferenças* e a *máquina analítica.*

A primeira delas, a *máquina de diferenças,* foi justamente idealizada para atender às necessidades da Marinha real inglesa. Na época, as tabelas de navegação eram escritas manualmente por diversos funcionários, contratados para:

1) realizar sucessivas e repetitivas operações de adição e multiplicação e
2) imprimir os resultados, escrevendo-os.

Foi constatado que, devido à natureza permanente e repetitiva do processo realizado por humanos, sempre ocorriam erros (tanto nos cálculos quanto na ocasião de registrar por escrito os resultados). O que Babbage se propunha (por contrato) era projetar uma máquina que realizasse de forma constante e sem erros o tedioso trabalho de cálculos e registrasse, de forma também confiável, os resultados.

A *máquina de diferenças* (assim chamada devido ao nome do processo matemático de cálculo utilizado por ela — diferenças finitas) era um dispositivo mecânico que só realizava adições e subtrações (como a máquina de Pascal) e cujos cálculos matemáticos se baseavam no processo de diferenças finitas, pelo qual é possível calcular fórmulas (até com polinômios e funções trigonométricas) utilizando apenas a operação de adição (a descrição detalhada do método e do trabalho de Babbage pode ser encontrada em [DUBB 78]). Ela era constituída de um conjunto de registradores mecânicos, cada um contendo rodas com dígitos, que serviam para armazenar um valor decimal. A exemplo da calculadora de Pascal, dois registradores adjacentes eram conectados por um mecanismo de efetuar somas. A máquina era acionada por um motor movido a vapor, que realizava uma série de etapas e finalmente apresentava um resultado. Além disso, ela continha um dispositivo de gravação em uma chapa de cobre, uma espécie de agulha que marcava os valores na chapa, a qual servia de matriz para posterior impressão em papel. Este processo de gravação pode ser considerado como pioneiro em termos de dispositivos de armazenamento secundário.

Babbage projetou algumas dessas máquinas, capazes de realizar cálculos com valores de até 15 algarismos e com polinômios de até 3.° grau. Ele conseguiu convencer o governo inglês a financiar a construção de uma máquina mais sofisticada e precisa, capaz de calcular polinômios de até 6.° grau e números de até 20 dígitos. Essa máquina funcionou e, após muito e muito dinheiro gasto, o governo inglês desistiu do projeto.

Nesse ínterim, Babbage passou a se dedicar ao projeto de um novo tipo de máquina, que se chamou *máquina analítica.*

A máquina era, na realidade, um computador mecânico capaz de armazenar 1000 números de 20 algarismos e que possuía um programa que podia modificar o funcionamento da máquina, fazendo-a realizar diferentes cálculos. Esta era sua grande diferença e vantagem sobre as anteriores, o fato de se tornar de uso mais geral por possuir a capacidade de modificar suas operações e assim realizar diferentes cálculos. Pode-se dizer que esta máquina foi a precursora dos primeiros computadores eletrônicos, inclusive no seu método de introduzir instruções por cartões perfurados (muito usados nas primeiras gerações de computadores eletrônicos).

Embora inteiramente mecânica, a máquina analítica de Charles Babbage essencialmente possuía os mesmos componentes que um computador atual:

- memória: constituída de rodas dentadas de contagem;
- processador: com uma unidade capaz de realizar as quatro operações aritméticas (operando com pares de registradores) e "unidade de controle", constituída de cartões perfurados convenientemente para realizar esta ou aquela operação.
- saída: para uma impressora ou para um dispositivo perfurador de cartões.

Além da fundamental característica de realização de programas de emprego geral, o projeto ainda acrescentou a capacidade de desvio da seqüência de ações da máquina, isto é, um prenúncio do que mais tarde (nos atuais computadores) seriam as instruções "Jump" (desvio incondicional) e "por exemplo, Jump on zero" (desvio condicional).

Alguns pesquisadores acreditam que Babbage utilizou em seus cartões perfurados a idéia de Joseph Jacquard, anteriormente citado.

O projeto final de Babbage, que pretendia ter a capacidade de calcular valores não com 20 dígitos, mas sim com 50 algarismos, nunca chegou a se tornar uma realidade física, talvez por estar realmente avançado demais para a época, quando a tecnologia de fabricação dos dispositivos necessários ao funcionamento das engrenagens não tinha a devida capacidade.

Alguns outros detalhes históricos interessantes podem ser citados, como a estimativa de Babbage de que sua máquina deveria realizar uma operação de adição em um segundo e uma multiplicação em um minuto, e que o programa criado para fazer a máquina funcionar foi desenvolvido por uma mulher chamada Ada Lovelace, que pode ser, então, considerada a primeira programadora de computador da história e que serviu de nome para a moderna linguagem de programação ADA, desenvolvida para o Departamento de Defesa dos EUA.

1.2.2 Época dos Dispositivos Eletromecânicos (1880-1930)

Com a invenção do motor elétrico, no fim do século XIX, surgiu uma grande quantidade de máquinas de somar acionadas por motores elétricos, baseados no princípio de funcionamento da máquina de Pascal. Essas máquinas se tornaram dispositivos comuns em qualquer escritório até o advento das modernas calculadoras de bolso em 1970.

Em 1889, Herman Hollerith desenvolveu o cartão perfurado para guardar dados (sempre o cartão perfurado, desde Joseph Jacquard) e também uma máquina tabuladora mecânica, acionada por um motor elétrico, que contava, classificava e ordenava informações armazenadas em cartões perfurados. Por causa dessa invenção, o Bureau of Census dos EUA contratou Hollerith em 1890 para utilizar sua máquina tabuladora na apuração de dados do censo de 1890. O censo foi apurado em dois anos e meio, apesar do aumento da população de 50 para 63 milhões de habitantes em relação ao censo de 1880, que consumiu quase 10 anos de processamento manual.

O sucesso de Hollerith com a apuração do censo conduziu à criação, em 1896, da Tabulating Machine Company, por onde Hollerith vendia uma linha de máquinas de tabulação com cartões perfurados. Em 1914, um banqueiro persuadiu três companhias a se juntarem, entre elas a empresa de Hollerith, formando a Computer Tabulating Recording Corporation. Thomas Watson foi contratado como gerente geral, e em 1924 ele mudou o nome da companhia para IBM — International Business Machines, logo após ter iniciado no Canadá uma bem-sucedida filial. Atualmente, os cartões perfurados não são mais utilizados (até a década de 1980, eles foram um dos principais elementos de entrada de dados dos computadores digitais, inclusive nos IBM/360/370 e de outros fabricantes). Também eram chamados de cartões Hollerith, assim como o código de 12 bits por eles usado também se denominava código Hollerith.

A primeira máquina de calcular eletrônica somente surgiu por volta de 1935, e seu inventor foi um estudante de engenharia alemão, Konrad Zuse, cuja idéia consistia em criar uma máquina que usava relés mecânicos que, atuando como chaves, podiam abrir ou fechar automaticamente, o que levou à utilização de números binários em vez de algarismos decimais, utilizados nas engrenagens da máquina de Babbage.

Em 1936, Zuse deixou de ser estudante e profissionalmente criou sua primeira máquina, chamada Z1, baseada em relés mecânicos, que usava um teclado como dispositivo de entrada e lâmpadas (dispositivo binário — acesa e apagada) como componente de saída (o primeiro microcomputador comercial, o Altair, em 1974 também usava lâmpadas como dispositivo de saída, embora ainda não empregasse o teclado como dispositivo de entrada). Zuse realizou alguns aperfeiçoamentos em seu "computador" até concluir, em 1941, o Z3, o qual utilizava relés eletromecânicos e era controlado por programa, sendo talvez o primeiro computador efetivamente operacional do mundo. Um outro modelo mais aperfeiçoado, o Z4, foi usado pelos militares alemães para auxiliar no projeto de aviões e mísseis durante a Segunda Guerra Mundial. Provavelmente Zuse teria desenvolvido máquinas de maior capacidade e versatilidade se tivesse sido mais bem financiado pelo governo alemão. Os bombardeios aliados na Alemanha destruíram a maior parte dos computadores construídos por Zuse, e por isso o seu trabalho foi praticamente perdido, restando apenas o registro histórico dessas invenções.

Outro "inventor" da época de dispositivos eletromecânicos foi Howard Aiken, um físico e matemático americano que desenvolveu um "computador", o Mark I, utilizando os princípios básicos da máquina de Babbage (era um sistema decimal e não binário, como os de Zuse), com engrenagens decimais e com estrutura computacional baseada em relés eletromecânicos. O projeto, que foi financiado pela IBM, era capaz de armazenar 72 números, e as instruções de dois operandos eram introduzidas na máquina por meio de uma fita de papel perfurado. Ao ser completado em 1944, o Mark I podia realizar uma soma em seis segundos e uma divisão em 12 segundos (Charles Babbage imaginava que sua máquina analítica poderia realizar uma adição em um segundo). No entanto, a eletrônica já começava a substituir elementos eletromecânicos por dispositivos muito mais rápidos, as válvulas, o que já tornava o Mark I obsoleto antes de operar comercialmente em escala, e o seu sucessor, o Mark II, nem chegou a ser concluído.

1.2.3 Época dos Componentes Eletrônicos — Primeiras Invenções (1930-1945)

O problema dos computadores mecânicos e eletromecânicos residia em dois fatos: *baixa velocidade* de processamento, devido à parte mecânica de seus elementos, e *falta de confiabilidade* dos resultados, já que seu armazenamento e movimento interno eram realizados por engrenagens, incapazes de realizar sempre o mesmo tipo de movimento, principalmente com o desgaste causado pelo tempo.

Esses dois problemas só poderiam ser solucionados com a utilização de elementos de armazenamento e chaveamento que não tivessem partes mecânicas e fossem bem mais rápidos. Para tanto, os cientistas dedicados a esse trabalho passaram a explorar o uso de um componente eletrônico, a válvula, inventada em 1906.

Pode-se dizer, *grosso modo*, que uma válvula é um dispositivo eletrônico, constituído de um tubo de vidro selado e que, em seu interior, a vácuo, ficam diversos elementos interligados de modo a permitir, de certa maneira, a passagem ou não de corrente elétrica. Esses elementos, catodo, anodo, grade e filamento, agem de maneira que o filamento produz aquecimento no catodo e anodo e, quando uma corrente elétrica é aplicada sobre eles, ela flui do catodo para o anodo devido à diferença de potencial entre eles. Quando se insere uma grade entre o catodo e o anodo, obtém-se um controle do fluxo da corrente, através da modificação da voltagem aplicada à grade (de valores negativos a positivos).

Quando se troca a voltagem sobre a grade acarreta a passagem ou não da corrente e, assim, a válvula age como se fosse uma chave com relação às placas. Desse modo, tem-se uma chave controlada eletronicamente (isto é, em alta velocidade), o que é muito mais eficaz do que um relé, controlado mecanicamente. Por essa razão, as válvulas passaram a ser utilizadas nos computadores substituindo os relés. Uma evolução considerável naquele tempo.

Na mesma época em que Zuse e Aiken realizavam seus trabalhos com dispositivos eletromecânicos, dois outros cientistas desenvolveram computadores usando válvulas.

Um desses cientistas foi John Vincent Atanasoff, que, por volta de 1939, projetou uma máquina calculadora para resolver equações lineares, mas a invenção apenas ficou registrada historicamente, sem que a intenção de seu inventor, de que a máquina se tornasse um dispositivo de emprego geral, fosse realizada. A grande importância dessa invenção foi, no entanto, a atenção que despertou em um outro cientista, John Mauchly, um dos construtores do computador, o ENIAC, que é atualmente reconhecido como o que deu início à computação eletrônica, como veremos logo adiante.

Além de Atanasoff, outro cientista, o matemático inglês Alan Turing, desenvolveu uma máquina com componentes eletrônicos. Turing é bastante conhecido pela teoria de computação que desenvolveu, conhecida como máquina de Turing, descrita em 1937, e que consistia na definição de uma função de computação, pela qual uma máquina poderia simular o comportamento de qualquer máquina usada para computação se fosse adequadamente instruída para tal (isto é, se recebesse instruções através de uma fita de papel perfurado). Porém, até pouco tempo atrás não havia registro de que ele tivesse desenvolvido outro tipo de trabalho, mais prático, para o desenvolvimento de computadores.

Recentemente, com a divulgação de documentos militares do governo britânico, antes sigilosos, é que se tomou conhecimento de que o primeiro computador verdadeiramente eletrônico foi colocado em operação em 1943, com o propósito de quebrar códigos militares secretos de comunicação dos alemães. Esta máquina, construída por Alan Turing com válvulas eletrônicas, foi denominada Colossus, provavelmente devido a seu tamanho. Sua grande desvantagem residia no fato de não ser uma máquina de emprego geral, pois não podia resolver outros problemas a não ser a quebra de códigos militares. Ela era, então, um sistema de computação com programa único.

1.2.4 A Evolução dos Computadores Eletrônicos (1945 — até quando?)

Apesar dos primeiros projetos de Atanasoff e Turing, reconhece-se outra máquina eletrônica como o primeiro computador, realmente falando, projetado como uma máquina de emprego geral, eletrônica e automática (!!!).

Costumava-se dividir (e não há razão para fazermos diferente) a evolução cronológica do desenvolvimento dos computadores até nossos dias de acordo com o elemento básico utilizado na fabricação dos componentes do processador central, o primeiro deles já citado como sendo a válvula eletrônica.

1.2.4.1 Primeira Geração: Computadores a Válvula

O primeiro computador eletrônico e digital, construído no mundo para emprego geral, isto é, com programa de instruções que podia alterar o tipo de cálculo a ser realizado com os dados, foi denominado ENIAC (Electronic Numerical Integrator And Computer) e foi projetado por John Mauchly e John P. Eckert, de 1943 a 1946, tendo funcionado daí em diante até 1955, quando foi desmontado.

Em agosto de 1942, na Universidade da Pensilvânia, John Mauchly, inspirado no projeto de Atanasoff, propôs ao Exército americano o financiamento para construção de uma máquina que auxiliasse os militares do Ballistics Research Laboratory (um departamento do exército americano responsável pela elaboração de tabelas de alcance e trajetória para novas armas balísticas) em seu trabalho, reduzindo o tempo de elaboração das tabelas balísticas. Na época, o laboratório empregava mais de 200 pessoas para o cálculo das tabelas, as quais, usando máquinas calculadoras de mesa, resolviam repetidamente equações balísticas para gerar os dados necessários à formação das tabelas. Tabelas para uma simples arma poderiam levar até dias para serem completadas, e isto atrasava consideravelmente a entrega dos artefatos.

O ENIAC era uma máquina gigantesca, contendo mais de 17.000 válvulas e 800 quilômetros de cabos. Pesava cerca de 30 toneladas e consumia uma enorme quantidade de eletricidade, além de válvulas, que queimavam com grande freqüência devido ao calor.

De qualquer modo, e apesar de ter ficado pronto após o término da guerra e, portanto, sem poder ser utilizado para o propósito inicial de seu financiamento, o ENIAC era extremamente rápido para sua época, realizando cerca de 10.000 operações por segundo. Ele possuía 20 registradores, cada um deles podendo armazenar um valor numérico de 10 dígitos; era uma máquina decimal (não binária) e, por isso, cada dígito era representado por um anel de 10 válvulas, uma das quais estava ligada em cada instante, indicando o algarismo desejado. O ENIAC era programado através da redistribuição de cabos em tomadas diferentes e rearranjo de chaves (possuía cerca de 6000), tarefa que poderia levar muitos dias (pode-se imaginar a redistribuição de cabos como uma tarefa análoga à das telefonistas em antigas mesas telefônicas).

O ENIAC provou, com sucesso, que era uma máquina de emprego geral ao ser utilizado para realização de complexos cálculos em relação ao uso da bomba H, uma tarefa bem diferente daquela para a qual ele tinha sido construído. No entanto, era uma máquina de difícil operação e de manutenção dispendiosa devido às sucessivas queimas de válvulas.

De qualquer modo, a divulgação das características do ENIAC despertou o interesse de numerosos cientistas da área, e vários projetos tiveram início na mesma época.

Enquanto Mauchly e Eckert iniciaram a construção de um novo computador, o EDVAC (Electronic Discret Variable Automatic Computer), um dos colaboradores do projeto ENIAC, o matemático John von Neumann também iniciou outro projeto de aperfeiçoamento do computador inicial, denominado IAS, nome do local onde von Neumann foi trabalhar, o Institute for Advanced Studies da Universidade de Princeton.

O EDVAC de Mauchly e Eckert não foi adiante devido à saída de ambos da Universidade de Pensilvânia, para constituírem sua própria empresa, que mais tarde se tornou a UNIVAC. Recentemente, a UNIVAC uniu-se à Burroughs, constituindo-se na atual Unysis Corporation.

A outra vertente do aperfeiçoamento do ENIAC, pelo desenvolvimento do EDVAC, é atribuída, como já mencionado, a John von Neumann, e é a ele que se credita de um modo geral a definição de uma arquitetura de computadores com *programa armazenado,* e que até os dias atuais é empregada nas máquinas modernas.

Em 1945, von Neumann divulgou seu conceito ao publicar a especificação básica do EDVAC, isto é, da sua versão do EDVAC, no trabalho "First Draft of a Report on the EDVAC" (Primeiro Rascunho de um Relatório sobre o EDVAC). O relatório definia as características essenciais de uma máquina seqüencial de programa armazenado. Nele foram introduzidos os aperfeiçoamentos desejados para reduzir os inconvenientes do ENIAC, tais como: a dificuldade de programar a recolocação da fiação (isto poderia ser realizado com o mesmo tipo de elementos que representavam os dados no ENIAC, eletronicamente) e o tipo de aritmética (substituindo a aritmética decimal pela binária devido à dificuldade e ao custo de construir uma máquina capaz de representar confiavelmente 10 níveis de tensão em vez de apenas 2).

Em 1946, von Neumann e vários outros cientistas em Princeton iniciaram a construção de uma nova máquina, um computador eletrônico de programa armazenado, o IAS, que se utilizava dos mesmos princípios descritos no referido relatório do EDVAC.

O IAS possuía as seguintes características básicas extraídas de [STAL 00] (embora pertença à primeira geração de computadores e tenha sido, para os padrões atuais, uma máquina limitada, o IAS é fundamental no estudo da arquitetura de computadores, pois a grande maioria de suas especificações permanece válida até o momento):

a) era constituído de quatro unidades principais (ver Fig. 1.6) — a memória, a UCP, a UC e a parte de entrada e saída;

b) possuía memória com 1000 posições, chamadas palavras, cada uma podendo armazenar um valor com 40 dígitos binários (bits) (ver Fig. 1.7);

c) tanto os dados (valores numéricos) quanto as instruções eram representados da mesma forma binária e armazenados na mesma memória;

d) possuía 21 instruções de 20 bits cada uma, constituídas de 2 campos, um com 8 bits, denominado código de operação (C.Op.), e o outro com 12 bits, denominado endereço, para localizar cada uma das 1000 palavras, endereços de 000 a 999 (embora pudesse endereçar 4096 (4K) posições de memória, pois $2^{12} = 4096$, o IAS somente possuía 1000 endereços);

e) operava de modo repetitivo, executando um *ciclo de instrução* em seguida ao outro. Cada ciclo consistia em dois subciclos: o ciclo *de busca* (*fetch cycle*), onde o C.Op. da próxima instrução era trazido da memória para o IR e a parte de endereço da instrução era armazenada no MAR (Memory Address Register). Tão logo o C.Op. estivesse armazenado no IR, então se iniciava o outro subciclo, o ciclo *de execução.* O circuito de controle interpretava o código de operação e gerava os sinais apropriados para acarretar o movimento de dados ou a realização de uma operação na UAL — Unidade Aritmética e Lógica.

Conforme pode ser observado dessas especificações resumidas, o IAS possuía características de arquitetura que permaneceram ao longo do tempo. As máquinas evoluíram consideravelmente em velocidade, capacidade de armazenamento, miniaturização, consumo de energia e calor, e outras inovações, mas a arquitetura básica permaneceu.

Em 1949, a empresa fundada por Mauchly e Eckert construiu com sucesso o primeiro computador para fins comerciais, o UNIVAC 1 (Universal Automatic Computer), adquirido pelo Bureau of Census dos EUA, para processar os dados do censo de 1950. Pouco mais tarde, a Mauchly-Eckert Computer Corporation foi absorvida pela Sperry-Rand Corporation, como uma de suas subsidiárias, com o nome de UNIVAC.

A UNIVAC fabricou diversos outros tipos de computadores, a começar pelo UNIVAC II e, em seguida, a série 1100, mais voltada para computação científica.

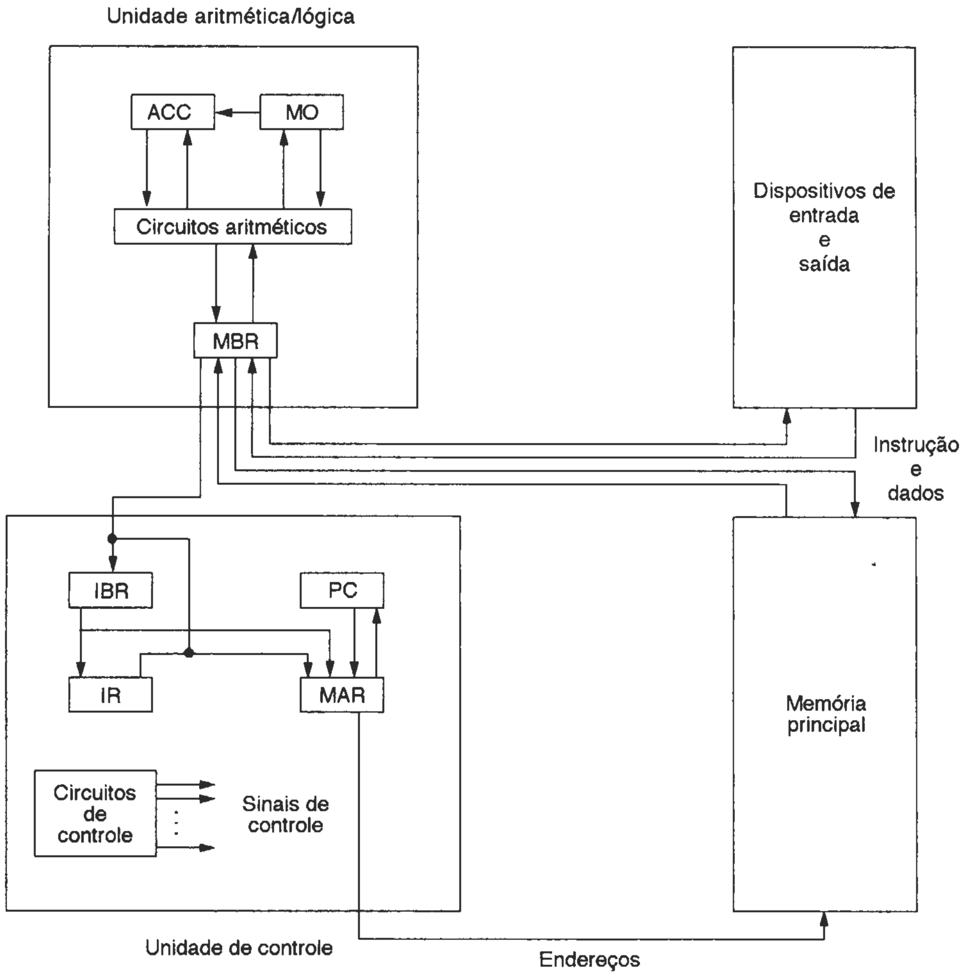
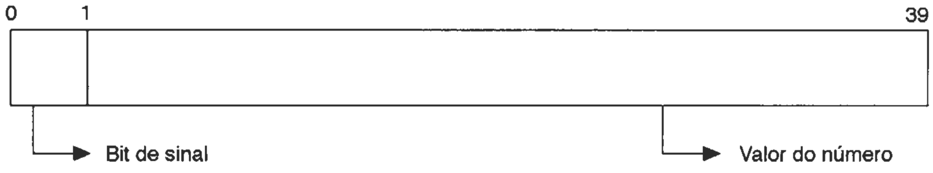
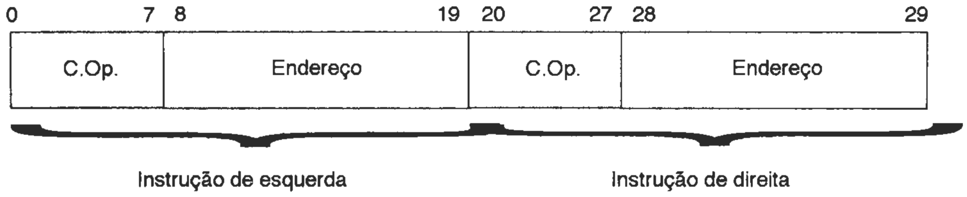

Unidade aritmética/lógica
ACC
MO
Circuitos aritméticos
MBR
Dispositivos de entrada e saída
Instrução e dados
IBR
PC
IR
MAR
Circuitos de controle
Sinais de controle
Memória principal
Unidade de controle
Endereços

Figura 1.6 Diagrama em bloco da estrutura do IAS.


0
1
39
Bit de sinal
Valor do número

(a) Representação de um dado em uma palavra


0
7
8
19
20
27
28
29
C.Op.
Endereço
C.Op.
Endereço
Instrução de esquerda
Instrução de direita

(b) Formato da palavra de instrução (duas instruções em uma palavra)

Figura 1.7 Formato de palavras de memória do IAS.

Em 1953, a IBM, até então mais voltada, e com sucesso, para a construção e comercialização de equipamentos de processamento por cartão perfurado, lançou o seu primeiro computador eletrônico de programa armazenado, o IBM-701, voltado para o processamento científico. Esta máquina possuía uma memória com 2K palavras de 36 bits. Em 1955, a IBM modificou o hardware do 701 para adaptá-lo ao uso comercial, lançando o IBM-702 e, em 1956, foi lançado o IBM-704, com 4K palavras de memória e, finalmente em 1958, a IBM lançou outra máquina, mais aperfeiçoada, o IBM-709. Nessa ocasião, a IBM já se destacava no mercado em relação à UNIVAC, que vinha sendo a n.º 1 desde 1950.

1.2.4.2 Segunda Geração — Computadores Transistorizados

A eletrônica moderna surgiu em 23 de dezembro de 1947, quando três cientistas do Bell Laboratories, John Bardeen, Walter Bratain e William Schockley, produziram pela primeira vez o *efeito transistor.* Eles descobriram que as propriedades condutoras de um diodo semicondutor poderiam ser controladas por um terceiro elemento. Os transistores se tornaram não só um sucesso em toda a indústria eletrônica (custo, tamanho e desempenho melhores que os dispositivos a válvula), mas também formaram a base de todos os computadores digitais até os dias atuais. O fato de que se pode ligar e desligar (dois estados) a corrente elétrica em um dispositivo é a base de toda a lógica digital (ver Cap. 4).

O transistor realiza as mesmas funções básicas de uma válvula, porém o faz consumindo muito menos energia e calor, o que o tornou rapidamente substituto completo das válvulas.

A primeira companhia a lançar comercialmente um computador transistorizado foi a NCR, e logo em seguida a RCA. As vantagens dessas máquinas sobre suas antecessoras a válvula eram várias: eram mais baratas, menores e dissipavam muito menos calor, além do menor consumo de energia elétrica.

Esta nova geração de computadores também teve, e muito, a participação ativa da IBM, já se firmando como a mais importante companhia na produção de máquinas científicas. Ela transformou a série 700 em série 7000, esta transistorizada. O primeiro deles, o 7090, e mais tarde o 7094, que possuía um ciclo de instrução de dois microssegundos e 32K palavras, ainda de 36 bits. Além do domínio na computação científica, a IBM também produziu uma máquina comercial de enorme sucesso, o IBM-1401 (quatorze zero um, como era conhecido).

Com esta geração de computadores, outros fatos historicamente importantes também aconteceram. Entre eles:

- O aparecimento de outra companhia fabricante de computadores, a DEC — Digital Equipment Corporation, que viria mais tarde a se tornar o segundo maior fabricante do mundo, depois da IBM. A DEC foi fundada em 1957 por Kenneth Olsen, um dos engenheiros do Lincoln Laboratory, do MIT (Massachusetts Institute of Technology), órgão que realmente desenvolveu o primeiro computador transistorizado, o TX-0 (apesar de o da NCR ter sido o primeiro do tipo comercial, o TX-0 foi o primeiro de todos, embora apenas em nível experimental). No mesmo ano de 1957, a DEC lançou seu primeiro computador, o PDP-1, início de uma longa série de máquinas extraordinariamente eficazes e tecnologicamente avançadas, até o famoso PDP-11. Por ser uma máquina de pequeno porte, comparada com os computadores de até então, o PDP-1 também custava muito menos. Por essa razão e devido ao excelente desempenho para a sua faixa de preço, o computador da DEC teve grande aceitação do mercado, tornando-se um marco inicial da indústria de minicomputadores, da qual a DEC foi líder por um longo período (primeiro com os PDP e, em seguida, com a família VAX).
- O aparecimento de unidades aritméticas e lógicas mais complexas, assim como unidades de controle.
- O aparecimento de linguagens de programação de nível superior ao das linguagens Assembly da época (na realidade, o FORTRAN para o IBM-704, em 1957, era ainda de primeira geração).
- O surgimento de outra companhia importante, a Control Data Corporation, que lançou, em 1964, o sistema CDC-6000, voltado primariamente para processamento científico (a CDC sempre construiu computadores com uma maior vocação para o processamento numérico). Era uma máquina com palavra de 60 bits (apesar de não ser múltipla da base 2, possuía um valor grande, apropriado para *processamento* numérico) e vários *processadores independentes* de entrada/saída, um total de 10, denominados PPU — Peripheral Processing Unit, que liberavam a UCP de várias tarefas, tornando o sistema ainda mais rápido.

1.2.4.3 Terceira Geração — Computadores com Circuitos Integrados

O desenvolvimento da tecnologia de circuitos integrados (Integrated Circuits — IC) surgiu devido à necessidade de se encontrar uma solução para os problemas de acomodação dos componentes eletrônicos (transistores, capacitores, resistores) nos equipamentos à medida que sua quantidade ia crescendo com o aumento da capacidade das máquinas. Das tentativas de encontrar solução para tais problemas é que se idealizou a possibilidade de acomodá-los em um único invólucro.

O ponto importante no conceito de circuitos integrados é que se pode formar múltiplos transistores em um único elemento de silício, de modo que, um circuito lógico que antes ocupava uma placa de circuito impresso completa pode ser, com esta tecnologia, acomodado em uma só pastilha (*chip*) de silício. E mais ainda, como se pode conectar vários transistores diretamente na pastilha, eles podem ser incrivelmente menores em tamanho, necessitando, assim, menos energia e dissipando menos calor.

Em outubro de 1958, Jack Kilby, da Texas Instruments Co., colocou dois circuitos em uma peça de germânio. O dispositivo resultante era rudimentar e as interconexões tinham que ser realizadas por fios externos, mas esse dispositivo é, em geral, reconhecido como o primeiro circuito integrado fabricado no mundo. Logo em seguida, Robert Noyce, da Fairchild Semiconductor Inc., utilizou-se de técnicas recém-criadas na mesma companhia e integrou múltiplos componentes em um substrato de silício. Os dispositivos comerciais que se sucederam mostraram a vantagem do silício sobre o germânio e permitiram o surgimento de uma nova geração de máquinas, mais poderosas e menores, devido à integração em larga escala (LSI — Large Scale Integration) que os circuitos integrados proporcionaram.

Em 1964, a IBM se utilizou das recentes inovações tecnológicas na área da *microeletrônica* (os circuitos integrados) e lançou a sua mais famosa "família" de computadores, a série/360. Este sistema incorporou diversas inovações, que se tornaram um marco histórico em termos de computação e consolidaram a posição já obtida pela IBM, como a primeira fabricante de computadores do mundo.

Entre essas inovações, podemos citar:

a) O conceito de família de computadores, em vez de máquina individual, como até então. Este conceito permite que o fabricante ofereça o mesmo tipo de máquina (arquitetura igual — linguagem de máquina semelhante etc.) com diferentes capacidades e preços, o que garante uma maior quantidade de clientes. O sistema/360 foi lançado inicialmente com cinco modelos, modelo 30, 40, 50, 65 e 75, cada um com características próprias de ciclo de instrução, capacidade de memória instalável, quantidade de processadores de E/S, embora todos os modelos tivessem o mesmo conjunto básico de instruções (e, com isso, um programa criado em um modelo poderia, em princípio, ser executado em outro). A Fig. 1.8 mostra um quadro comparativo entre os diversos modelos de família.

b) A utilização de unidade de controle com microprogramação em vez das tradicionais unidades de controle no hardware (ver Cap. 6).

Características da família/360					
Características	Modelo 30	Modelo 40	Modelo 50	Modelo 65	Modelo 75
Capacidade máxima de MP (bytes)	64K	256K	256K	512K	512K
Ciclo do processo em microssegundos	1	0,625	0,5	0,25	0,2
Quantidade máxima de canais (E/S)	3	3	4	6	6
Bytes puxados da MP por ciclo	1	2	4	16	16

Figura 1.8 Características principais da família IBM/360.

c) O emprego de uma técnica chamada multiprogramação, pela qual vários programas compartilham a mesma memória principal e dividem o uso da UCP, dando a impressão ao usuário de que estão sendo executados simultaneamente.

d) A elevada capacidade de processamento (para a época), com palavra de 32 bits e ciclo de instrução de até 250 nanossegundos, bem como a grande capacidade de armazenamento na memória principal, 16 Mbytes.

e) Memória principal orientada a byte, isto é, cada célula de MP armazena oito bits de informação, independentemente do tamanho de bits definido para a palavra de dados. Esta característica tornou-se comum para quase todo o mercado (exceto máquinas científicas) e até hoje os computadores continuam com a MP orientada a byte.

f) O lançamento de um programa (conjunto de programas é o melhor termo) gerenciador dos recursos de hardware, de modo mais integrado e eficaz, o sistema operacional OS/360.

Além da família /360, esta época de LSI presenciou também o lançamento de outro minicomputador DEC, com circuitos integrados, memória principal orientada a byte e palavra de 16 bits, o PDP-11, uma das máquinas mais famosas em sua categoria. Seu sucessor, o sistema VAX-11, também teve o mesmo sucesso, especialmente no ambiente universitário.

1.2.4.4 Quarta Geração — Computadores que Utilizam VLSI

O termo VLSI (Very Large Scale Integration), integração em larga escala, caracteriza uma classe de dispositivos eletrônicos capazes de armazenar, em um único invólucro, milhares e até milhões de diminutos componentes. Este dispositivo, já anteriormente mencionado e denominado pastilha (*chip*), vem constituindo a base da estrutura de todos os principais sistemas de computação modernos (ver item 4.3).

A técnica de miniaturização de componentes eletrônicos ou microeletrônica conduziu, por volta de 1971/1972, ao desenvolvimento de um outro tipo de computadores até então inexistente no mercado — os computadores pessoais ou microcomputadores.

No Cap. 2 é apresentada uma classificação usualmente aceita de cinco categorias de computadores, definidas em função de algumas de suas características básicas, como velocidade de processamento, tipicamente medida em MIPS (milhões de instruções por segundo) ou MFLOPS (milhões de operações de ponto flutuante por segundo) (ver descrição sobre aritmética em ponto flutuante no Cap. 7), custo e tamanho físico (esta é uma característica bastante controvertida em face da capacidade de miniaturização atual das máquinas).

A evolução dos microcomputadores, decorrente principalmente do avanço na miniaturização dos processadores e demais elementos, vem sendo de tal forma rápida e eficiente que os computadores de maior porte foram sendo progressivamente substituídos nas empresas, restando hoje um nicho de mercado bem pequeno e específico para aquelas máquinas. Atualmente, pode-se afirmar que a maioria dos sistemas de computação utilizados no mundo comercial e governamental é baseada em microcomputadores, assim como o imenso universo dos computadores pessoais.

Vamos nesse ponto escolher alguns tópicos principais, característicos da época atual, para mencionar as observações mais interessantes, principalmente para o leitor iniciante. Alguns dos pontos adiante abordados têm apenas interesse histórico, mas parecem relevantes para o conhecimento da evolução da computação.

1.2.4.5 Evolução dos Computadores de Grande Porte (Mainframes)

Os computadores de grande porte se constituíram nas principais máquinas das empresas, desde os primórdios da computação, com o lançamento do UNIVAC 1 e do MM-701, e até alguns anos atrás, quando a capacidade sempre crescente e o custo bem menor dos microcomputadores orientaram as intenções dos usuários na ocasião de implantar novos sistemas ou atualizar os antigos, grande parte deles substituindo os mainframes por estações de trabalho em rede ou mesmo redes locais de microcomputadores.

Um dos principais representantes dessa categoria, o sistema IBM/360, teve uma evolução tecnológica acentuada e permanente, desde 1964 até 1988, já com o nome de /370, tendo continuado o desenvolvimento com novos sistemas, porém sempre com a mesma arquitetura básica, como os MM-43xx, IBM-308x e IBM-309x.

A Fig. 1.9 apresenta um quadro demonstrativo da evolução dos sistemas /360 e /370, incluindo as principais inovações de cada família.

Outra classe de computadores bastante específica e com aplicações científicas definidas é a de supercomputadores, entre os mais significativos estão a família CRAY (CRAY-1, CRAY-2, CRAY-X/MP, CRAY-Y/MP), a família WM-90xx, com processamento vetorial, e a família CDC-CYBER.

Ano	Evento Principal
1964	Lançamento do Sistema /360 pela IBM
1968	Representação de Números em Ponto Flutuante
	Arredondamento
	Alinhamento a Nível de Byte
	E/S Multiplexada a Bloco (Bloco Multiplex I/O)
1970	Lançamento do Sistema /370 pela IBM
	Relógio (Time-of-day clock)
	Registradores de Controle
	Seis Novas Instruções de Emprego Geral
1972	Grandes Alterações que Diferenciam o Sistema /360 do Sistema /370
	Memória Virtual
	Modo de Controle Estendido
	Registro de Eventos de um Programa
	Timer da UCP
	Comparador entre Valores de Relógio (clock comparator)
1973	Extensões para Multiprocessamento
	Instruções de Manuseamento de PSW
	Instruções que tratam de troca de programas na forma condicional (conditional swapping)
1978	Proteção de Endereços de Memória
1979	Chaveamento de Conjunto de Canais
1981	Facilidades de Manipulação de Duplo Espaço de Endereçamento
	Proteção de Segmentos
	Instruções de Enfileiramento de E/S
1983	Lançamento do Sistema /370 pela IBM - Arquitetura Estendida
	Endereçamento de 31 bits
	Novo Subsistema de Canal de E/S
	Proteção de Páginas
1984	Execução Interpretativa
1986	Lançamento do Sistema /370 - Facilidade Vetorial pela IBM
	Registradores de Vetores
	Instruções de Manipulação de Vetores
1988	Lançamento da Arquitetura /370 - Sistema Enterprise pela IBM
	Registradores de Acesso
	Modo de Endereçamento para Acesso a Registrador
	Pilha de Ligação

Figura 1.9 Quadro demonstrativo da evolução dos sistemas/360 e /370.

1.2.4.6 Computadores Pessoais — Microcomputadores

Em 1971, uma companhia criada para produzir componentes eletrônicos, a Intel Corporation, sediada na Califórnia, EUA, produziu a primeira UCP em uma só pastilha de circuito integrado, denominada INTEL-4004. Esta UCP, que se destinava a uma calculadora, possuía palavra de 4 bits e tinha cerca de 2300 transistores na pastilha. Logo em seguida, a Intel lançou um novo microprocessador, desta vez com 8 bits de palavra e 16K de memória, o Intel 8008.

Tanto o 4004 quanto o 8008 eram UCP destinadas a uma aplicação específica (o 8008 destinava-se à Display Terminals Corporation, para servir de controlador de um monitor de vídeo). Embora a empresa solicitante da pastilha nunca tivesse usado o 8008, a Intel vendeu uma quantidade não esperada dessa pastilha, mesmo com os problemas de pouca memória e pequeno conjunto de instruções. Então, em 1973, a Intel lançou o seu grande sucesso da época, o primeiro microprocessador de emprego geral do mundo, o Intel 8080. O 8008 possuía cerca de 3500 transistores encapsulados na pastilha, enquanto o 8080 tinha em torno de 5000 transistores. Este último possuía também 8 bits de tamanho de palavra, capacidade maior de memória (cada endereço tinha 16 bits e, então, a memória podia conter até 64 Kbytes) e um grande conjunto de instruções (78 instruções). O 8080 vendeu aos milhões e, desde então, a Intel não parou mais de crescer e desenvolver novos produtos, até o seu mais recente lançamento, o microprocessador Pentium, contendo cerca de 3,5 milhões de transistores na pastilha.

Na realidade, os computadores pessoais surgiram com o lançamento do Altair, que pode ser considerado o primeiro computador pessoal oferecido com fins comerciais, auxiliando sobremodo o início da revolução que os microcomputadores realizaram desde então. Esse microcomputador, construído pela empresa MITS, baseava-se no microprocessador Intel 8080 e utilizava um interpretador da linguagem, Basic, desenvolvido por Bill Gates e Paul Allen, que fundaram nessa ocasião a Microsoft, tornando-a o gigante atual. O Altair foi um verdadeiro sucesso comercial. A Fig. 1.10 apresenta o Altair ao lado de um moderno Laptop e a Fig. 1.11 mostra o Altair de frente, com suas chaves e lâmpadas (dispositivos de entrada e saída).

Desde o surgimento dos primeiros microprocessadores fabricados pela Intel até os dias atuais, dos Pentium, da mesma Intel, aos K6, K7 e Athlon, da AMD (Advanced Micro Devices), aos SPARC, da Sun Microsystems e outros, a evolução da microeletrônica e da tecnologia de fabricação e montagem de componentes e equipamentos completos tem sido constante e extraordinariamente rápida. A quantidade de inovações e marcos nesse desenvolvimento é enorme, conteúdo de livros inteiros.

A Tabela 1.1 apresenta um quadro demonstrativo com dados de alguns dos principais microprocessadores surgidos desde o Intel 4004, de modo a permitir ao leitor conhecer alguns desses dispositivos. No Cap. 6 serão apresentados mais detalhes de suas funções e características.

Por outro lado, apresenta-se na Tabela 1.2 um quadro demonstrativo da evolução da ciência da computação, incluindo-se naturalmente algumas das observações inseridas neste item.

Para encerrar este texto devemos mencionar algumas observações sobre outros fabricantes, além da Intel, que têm contribuído sobremaneira para o aperfeiçoamento e crescimento do mercado da microcomputação.

A Motorola é um desses fabricantes que, a partir do lançamento de seu processador MC6800 de 8 bits, em 1974, também não parou de evoluir. Em 1979 a Motorola inovou, lançando o microprocessador MC68000, já de 32 bits de palavra (a Intel lançou seu primeiro processador de 32 bits, o Intel 80386, somente em 1985,

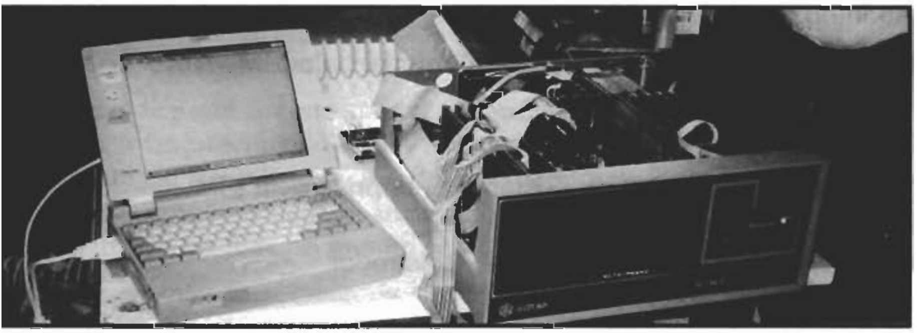


Figura 1.10 O microcomputador Altair ao lado de um moderno notebook.

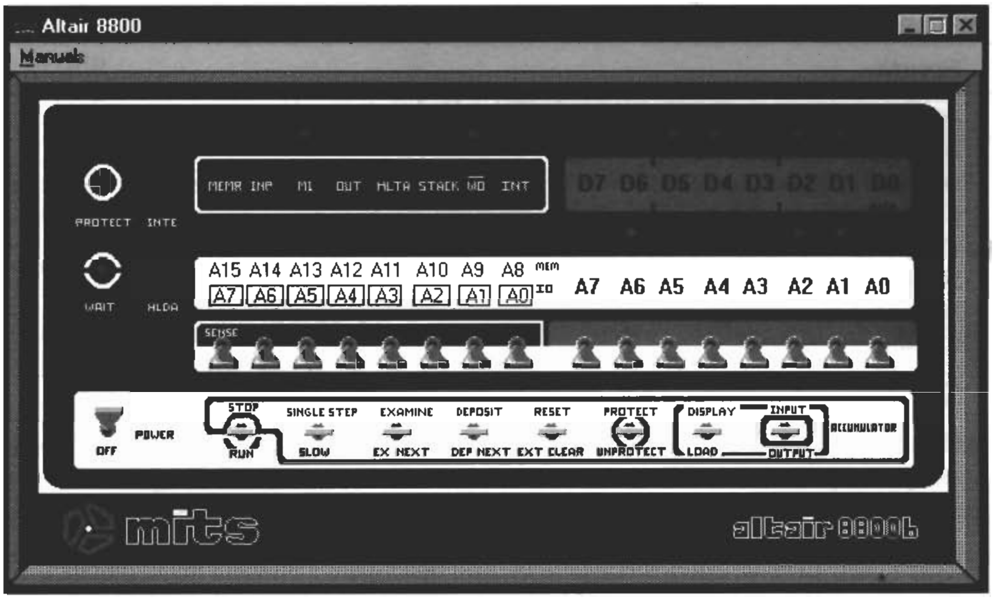

Altair 8800
Manuals
PROTECT
INTE
MEMR INP M1 OUT HLTA STACK WO INT
D7 D6 D5 D4 D3 D2 D1 D0
WAIT
HLDA
A15 A14 A13 A12 A11 A10 A9 A8
A7 A6 A5 A4 A3 A2 A1 A0
MEM
IO
A7 A6 A5 A4 A3 A2 A1 A0
SENSE
OFF
POWER
STOP
RUN
SINGLE STEP
SLOW
EXAMINE
EX NEXT
DEPOSIT
DEP NEXT
RESET
EXT CLEAR
PROTECT
UNPROTECT
DISPLAY
LOAD
INPUT
OUTPUT
ACCUMULATOR
mits
altair 8800b

Figura 1.11 O microcomputador Altair.

seis anos depois), mas não conquistou o mercado, como era esperado. Alguns dos principais processadores Motorola constam da Tabela 1.1.

Com o passar do tempo, o mercado de computadores pessoais se constituiu de equipamentos fabricados em torno desses dois principais fabricantes: a Intel, com a fatia maior do mercado, acionando todos os microcomputadores do tipo PC, e a Motorola, como base dos computadores Macintosh (da Apple) e Amiga (da Comodore).

Tabela 1.1 Quadro Demonstrativo da Evolução de Microprocessadores

Microprocessadores	Data de lançamento	Palavra de dados	Endereçamento máximo
Intel 4004	1971	4	1 Kbyte
Intel 8080	1973	8	64 Kbytes
Intel 8088	1980	16	1 Mbyte
Intel 80286	1982	16	16 Mbytes
Intel 80386	1985	32	4 Gbytes
Intel 80486	1989	32	4 Gbytes
Intel Pentium I	1993	32	4 Gbytes
Intel Pentium Pro	1995	32	64 Gbytes
Intel Pentium II	1997	32	64 Gbytes
Intel Pentium III	1999	32	64 Gbytes
Motorola 6800	1974	8	64 Kbytes
Motorola 68000	1979	32	16 Mbytes
Motorola 68010	1983	32	16 Mbytes
Motorola 68020	1984	32	4 Gbytes
Motorola 68030	1987	32	4 Gbytes
Motorola 68040	1989	32	4 Gbytes
Zilog Z-80	1974	8	64 Kbytes
Zilog Z-8000	1979	16	1 Mbyte
AMD-K6	1997	32	4 Gbytes
AMD-K6-2	1998	32	
AMD Athlon	1999	32	64 Gbytes
Cyrix 6x86MX	1997	32	
Cyrix MII	1998	32	

Tabela 1.2 Eventos Relevantes da Evolução da Computação

Período	Evento
500 a.C.	Invenção e utilização do ábaco
1642 d.C.	Blaise Pascal cria sua máquina de somar.
1670	Gottfried Leibniz cria uma máquina de calcular que realiza as quatro operações aritméticas.
1823	Charles Babbage cria a máquina de diferenças, por contrato com a Marinha Real Inglesa.
1842	O mesmo Babbage projeta uma máquina analítica para realizar cálculos.
1889	Herman Hollerith inventa o cartão perfurado.
1890	Hollerith desenvolve um sistema para registrar e processar os dados do censo.
1924	Constituição da IBM.
1939	John Atanasoff projeta o primeiro computador digital.
1946	Término da construção do ENIAC.
1946	John von Neumann propõe que um programa seja armazenado no computador e projeta o IAS, implementando sua proposta.
1951	Termina a construção do primeiro computador comercial de propósito geral, o UNIVAC.
1956	Termina a montagem do primeiro computador transistorizado, o TX-0, no MIT.
1957	Uma equipe da IBM, liderada por John Bachus, desenvolve a primeira linguagem de alto nível, Fortran, voltada para solucionar problemas matemáticos.
1958	A IBM lança o IBM-7090.
1958	Jack Kilby, na Texas Instruments, completa a construção do primeiro circuito integrado, contendo cinco componentes.
1962	Douglas Engelbart, do Stanford Research Institute, inventa o mouse.
1964	A IBM lança o IBM/360, primeiro computador a utilizar circuitos integrados.
1964	A linguagem Basic (Beginners All-purpose Symbolic Instruction Code) é desenvolvida por Thomas Kurtz e John Kennedy no Dartmouth College. Mais tarde, ela se torna popular devido ao lançamento do Altair com o interpretador desenvolvido por Bill Gates e Paul Allen, fundadores da Microsoft.
1965	Gordon Moore, diretor de pesquisa e desenvolvimento da empresa Fairchild Semiconductor, prevê que a densidade dos transistores e circuitos integrados dobraria a cada 12 meses nos 10 anos seguintes. Esta previsão foi atualizada em 1975, substituindo 12 meses por 18 meses e tornou-se conhecida como Lei de Moore.
1965	A IBM fabrica o primeiro *floppy disk*.
1967	A primeira versão do sistema operacional Unix é lançada, rodando em um computador DEC PDP-7. Este sistema foi escrito, a partir de 1969, no Bell Laboratories, por Dennis Ritchie e Ken Thompson.
1970	A linguagem Pascal é projetada por Nicklaus Wirth.
1971	A Intel lança o primeiro sistema de microcomputador, baseado no processador 4004, com desempenho de 60.000 operações por segundo e 2.300 transistores encapsulados.
1971	Dennis Ritchie, do Bell Labs, desenvolve a linguagem C.
1972	Gary Kildall escreve um sistema operacional na linguagem PL/M e o denomina CP/M (Control Program/Monitor).
1973	A Intel lança o processador 8080 de 2 MHz (primeiro lançamento em 1973), com 6000 transistores e 640.000 instruções por segundo. O CP/M é adaptado para o 8080 e a Motorola lança seu processador de 8 bits, o 6800.
1974	Na edição de janeiro da revista *Popular Electronics* é realizado o lançamento do primeiro microcomputador de 8 bits, o Altair.
1975	Steve Wozniak e Steve Job formam a Apple Computer. A DEC lança um de seus mais populares minicomputadores, o VAX 11/780.
1976	A Apple Company lança seu computador Apple II.
1976	Surge a primeira planilha eletrônica, Visicalc.
1977	A IBM anuncia o lançamento de seu primeiro microcomputador, o IBM-PC.
1979	A Apple apresenta seu primeiro computador do tipo Macintosh.
1981	A Microsoft lança sua planilha Excel, o primeiro aplicativo para o Windows.
1984	A Microsoft lança seu sistema operacional Windows para IBM-PCs.
1987	A Microsoft lança a versão 3.0 do Windows para PCs.

Tabela 1.2 Eventos Relevantes da Evolução da Computação (continuação)

Período	Evento
1989	A AMD lança seu clone do processador Intel 386.
1990	A IBM e a Motorola estabelecem um acordo para desenvolvimento do microprocessador PowerPC.
1991	A IBM lança um microcomputador portátil, o ThinkPad 700C.
1992	Linus Torvalds desenvolve o sistema operacional Linux, na Finlândia.
	A NCSA desenvolve o primeiro navegador para Internet, o Mosaic.
1993	O Mosaic se transforma no Netscape.
1994	A Iomega lança seus Zip drives.
1995	Inicia-se o contencioso entre a Microsoft e o governo dos EUA, que dura até os dias atuais.
1996	Aparecem no mercado os CD-RW (CD que podem ser regravados).
1998	A Compaq adquire a DEC.

Outro fabricante de processadores e demais componentes para o mercado de microcomputadores é a AMD — Advanced Micro Devices, criada em 1974 para concorrer diretamente com a Intel, sendo fabricante dos atuais processadores K-7 e Athlon, integrantes de uma quantidade apreciável de microcomputadores no mundo inteiro.

Além da AMD, pode-se mencionar como empresas relevantes na área de equipamentos de computação a Sun Microsystems, a Compaq, a Dell e a Gateway.

Antes de concluir este tópico, é interessante também citar um outro fabricante de microprocessadores, a Zilog, que foi constituída pelo pessoal que saiu da Intel na época, fabricante do Z-80, processador de 8 bits (1974) e do Z-8000 (de 16 bits). A Zilog deixou o mercado de microcomputadores de emprego geral e atualmente se destina a fabricar microprocessadores para controle de processos.

EXERCÍCIOS

1) Conceitue os termos *dado* e *informação*, no que se refere a seu emprego em processamento de dados.

2) Caracterize as etapas principais de um processamento de dados.

3) Conceitue um sistema. Cite dois exemplos práticos de organizações sistêmicas na vida real.

4) Considerando a organização de sistemas de informação definida no item 1.1.2, cite exemplos práticos de sistemas de nível operacional, gerencial e de alto nível (estratégicos).

5) O que você entende por um programa de computador?

6) Conceitue os termos *hardware* e *software*.

7) O que é e para que serve uma linguagem de programação de computador? Cite exemplos de linguagens de programação.

8) Quem desenvolveu a máquina analítica?

9) Qual foi a característica marcante do censo de 1890 dos EUA, no que se refere à contabilização dos dados levantados?

10) Qual foi o propósito que conduziu ao desenvolvimento do primeiro computador eletrônico do mundo?

11) Qual foi o primeiro microprocessador de 8 bits lançado comercialmente? Qual o nome da empresa proprietária?

12) Quais eram as características básicas da arquitetura proposta por John von Neumann?

2

Componentes de um Sistema de Computação

2.1 DESCRIÇÃO DOS COMPONENTES

No capítulo anterior, vimos que um sistema de computação é um conjunto de componentes integrados para funcionar como se fossem um único elemento e que têm por objetivo realizar manipulações com dados, isto é, realizar algum tipo de operações com os dados de modo a obter uma informação útil.

Como mencionamos um conjunto de componentes, vamos procurar identificar melhor cada um deles, utilizando para isso um exemplo corriqueiro. Na realidade, a finalidade básica deste texto consiste em apresentar, individualmente, os componentes principais que constituem um sistema de computação, o computador. Esta apresentação compreenderá a descrição funcional do referido componente, a memória ou o processador central, por exemplo, bem como pretende mostrar exemplos práticos dos componentes reais atualmente fabricados, de modo que o leitor tenha uma idéia viva sobre o que se está descrevendo teoricamente.

Consideremos o caso de um sistema de controle do movimento diário de uma agência bancária, no que se refere exclusivamente à atualização dos saldos das contas de clientes que tiveram movimento em um determinado dia. Em linhas gerais (e de forma bem simplificada, apenas com o propósito já mencionado de procurarmos identificar os principais componentes de um sistema de computação), o movimento do dia compreenderia apenas retiradas de algum valor (por exemplo, com um cheque, cartão etc.) ou inclusões (como depósito por cheques, cartões, espécie etc.). As duas possíveis operações seriam, então, **retirada** — operação de subtrair do saldo atual o valor da retirada, obtendo-se um novo valor de saldo, e **depósito** — operação de somar ao saldo atual o valor do depósito, obtendo-se um novo valor de saldo. Ambas as operações são realizadas por meio de informações obtidas de um documento — DOC — que contém o número da conta a ser manipulada, o tipo da operação (retirada ou depósito) e o valor em moeda.

A Fig. 2.1(a) mostra o processo de atualização através da descrição, em linguagem clara, sem qualquer compromisso de identificá-la com alguma linguagem de programação, das etapas (tarefas) a serem realizadas para a referida atualização.

A relação de tarefas que descrevemos em linguagem clara na Fig. 2.1(a) é denominada algoritmo (ver definição de algoritmo no item 1.1.3). No entanto, um algoritmo descrito do modo informal como o fizemos na figura não consegue ser processado por uma máquina, justamente devido à sua informalidade e à ausência de qualquer padrão de nomenclatura. Cada pessoa pode escrever a mesma relação de tarefas, porém usando palavras e frases ligeiramente diferentes. Isso impede que uma máquina entenda que tarefa deve ser realizada (por exemplo, alguém pode denominar "adicionar" a operação de somar. Como uma máquina deve entender esta operação?). Por isso, foi necessário definir linguagens de comunicação com os computadores, denominadas, de forma genérica, linguagens de programação (ver item 1.1.3). Nesse nosso exemplo, então, o passo

```
Início do Programa
Enquanto houver DOC

Fazer
    • Obter um DOC
    • Ler número do DOC
    • Encontrar conta com número = número do DOC
    • Se tipo-DOC = depósito
          Então: Novo-saldo = Saldo + Valor
    • Se tipo-DOC = retirada
          Então: Novo-saldo = Saldo - Valor
    • Escrever Novo-saldo no lugar de Saldo
Fim do Fazer
Fim do Programa
```

Figura 2.1(a) Exemplo de um algoritmo simplificado para atualização de saldo de contas bancárias.

seguinte seria codificar o algoritmo em comandos de uma linguagem de programação de alto nível do tipo Pascal ou ainda C, Visual C, Delphi.

Em seguida, os comandos definidos — que se constituem, em conjunto, no que se denomina um Programa de Computador — precisam ser interpretados pela máquina (pelo computador) e, para tal, precisam de algum modo ser introduzidos no hardware. Trata-se da primeira etapa de um processamento (ver item 1.1.1) — Entrada — que requer um componente ou equipamento específico (hardware). Por exemplo, podemos digitar caractere por caractere do programa em questão, usando um componente denominado Teclado, bastante semelhante ao teclado das máquinas de escrever comuns. No item 10.3.1 é descrito o funcionamento de um teclado, mostrado na Fig. 10.16. Há inúmeros outros equipamentos que podem ser utilizados como componente de entrada de dados em um sistema de computação, como, por exemplo, um mouse, uma unidade de disco magnético (como os disquetes, tão populares em microcomputadores), um sensor ótico (utilizado para "ler" as marcas a lápis, colocadas em folhas de respostas por candidatos em um vestibular) ou um sensor magnético (utilizado para "ler" marcas colocadas em um cheque de banco). (Ver Cap. 10 — Entrada e Saída, para uma descrição mais detalhada de diversos componentes de entrada de um sistema de computação.)

No entanto, os computadores foram (e ainda são) projetados com capacidade de entender e realizar apenas tarefas bem simples e curtas, tais como: somar dois números de cada vez (eles não efetuam operações com três ou quatro números de uma só vez etc.), mover um número de um local para outro, ler o caractere correspondente à tecla que acabamos de pressionar no teclado e assim por diante. Então, para que o computador possa realizar as tarefas que relacionamos de modo geral na Fig. 2.1(a), precisamos detalhá-las mais, de modo que as novas tarefas sejam iguais às operações que o hardware sabe fazer. Ou seja, o programa introduzido no sistema pelo dispositivo de entrada não pode ser diretamente processado, pois seus comandos são complexos para o entendimento da máquina. E, por isso, foi preparada uma nova relação com outras tarefas, mais detalhadas e simples, que produzem, porém, o mesmo resultado final, solucionando o mesmo problema. A Fig. 2.1(b) mostra esta nova relação (programa), que compreende as operações que o hardware pode realizar. Na realidade, algumas das instruções relacionadas na figura precisam ainda ser mais detalhadas, mas trata-se de situação bem específica (de entrada e saída), a ser discutida no Cap. 10.

Neste momento, vamos ignorar como foi realizada a transformação do programa da Fig. 2.1(a) no programa da Fig. 2.1(b) (ver Cap. 9), como também nada sabemos ainda sobre o formato dos elementos que constituem os referidos programas. Queremos, neste instante, tão-somente identificar quais são os componentes envolvidos com a realização das tarefas descritas nas figuras citadas e conhecer suas funções básicas dentro do processo global.

Retornando ao algoritmo da Fig. 2.1(b), devemos ter atenção ao fato de que, para que uma máquina seja capaz de realizar várias operações, é preciso que ela seja de algum modo instruída para identificar cada uma

```
Início do Programa
INÍCIO  Obter DOC
        Se não há mais DOC,
           Então: Vá para FIM
           Senão: Fazer 1:
              Ler número do DOC
CONTA         Obter Nova Conta
              Ler Número da Conta
              Se Número do DOC = Número da Conta
                 Então: Fazer 2:
                        Ler tipo do DOC
                        Se tipo do DOC = Depósito
                           Então: Trazer Valor da Conta para Calculador
                                  Trazer Valor do DOC para Calculador
                                  Somar: Valor da Conta + Valor do DOC = Resultado
                                  Substituir Valor da Conta por Resultado
                           Senão: Trazer Valor da Conta para Calculador
                                  Trazer Valor do DOC para Calculador
                                  Subtrair: Valor da Conta - Valor do DOC = Resultado
                                  Substituir Valor daConta por Resultado
                        Fim de Fazer 2
                 Senão: Retornar para CONTA
           Fim de Fazer 1
FIM     Fim do Programa
```

Figura 2.1(b) Descrição mais detalhada do algoritmo da Fig. 2.1(a).

delas e, depois de identificá-la, saber como realizá-la. As tarefas relacionadas na Fig. 2.1(b) são, uma por uma, operações que uma determinada máquina (o hardware) pode realizar. Chamam-se por causa disso *instruções de máquina*. O componente do computador que é capaz de entender e realizar uma operação definida por uma instrução de máquina denomina-se Unidade Central de Processamento — UCP, ou simplesmente *processador central* (CPU — Central Processing Unit). Uma UCP ou processador é constituída de milhões de minúsculos circuitos e componentes eletrônicos (transistores, resistores etc.), cujas funções básicas são ler e interpretar instruções de máquina e realizar as operações matemáticas (ou outras) definidas após a interpretação de uma determinada instrução (ver Cap. 6). Atualmente, os mencionados milhões de elementos podem ser encapsulados em um único invólucro, formando as pastilhas (*chips*), que já citamos no Cap. 1, como os processadores (UCP) Intel 80486, Intel Pentium, AMD K6, AMD K7, Motorola 68040, Power PC (IBM/Motorola/Apple) e outras.

Para que a UCP possa trabalhar — entender e executar uma instrução de máquina — é necessário, em primeiro lugar, que o programa mostrado na Fig. 2.1(a) seja introduzido no sistema (através de um dispositivo de entrada, como o teclado), para em seguida ser convertido no programa da Fig. 2.1(b), e depois a UCP começar a executar este último.

Já vimos anteriormente que um programa é sempre constituído de várias instruções e para que ele seja rapidamente executado, é necessário que ele execute todas as instruções, recebendo os dados, manipulando-os e expondo todos os resultados, de forma totalmente automática. Em outras palavras, antes da execução do programa, este e os dados que serão por ele manipulados, devem ser armazenados na própria máquina para, um a um, serem localizados pelo processador, entendidos e executados, sem que haja interveniência de uma pessoa (pois, nesse caso, haveria sempre um tempo de atraso bem grande). O componente do sistema de computação responsável pelo armazenamento das informações introduzidas pelo componente de ENTRADA é denominado Memória (ver Fig. 2.2).

Após a realização de todas as operações, os resultados devem ser apresentados ao usuário interessado, o qual naturalmente deseja vê-los em uma forma inteligível para ele (caracteres alfabéticos, algarismos decimais, sinais de pontuação da nossa linguagem etc.) e não na linguagem do computador. Esses resultados podem ser

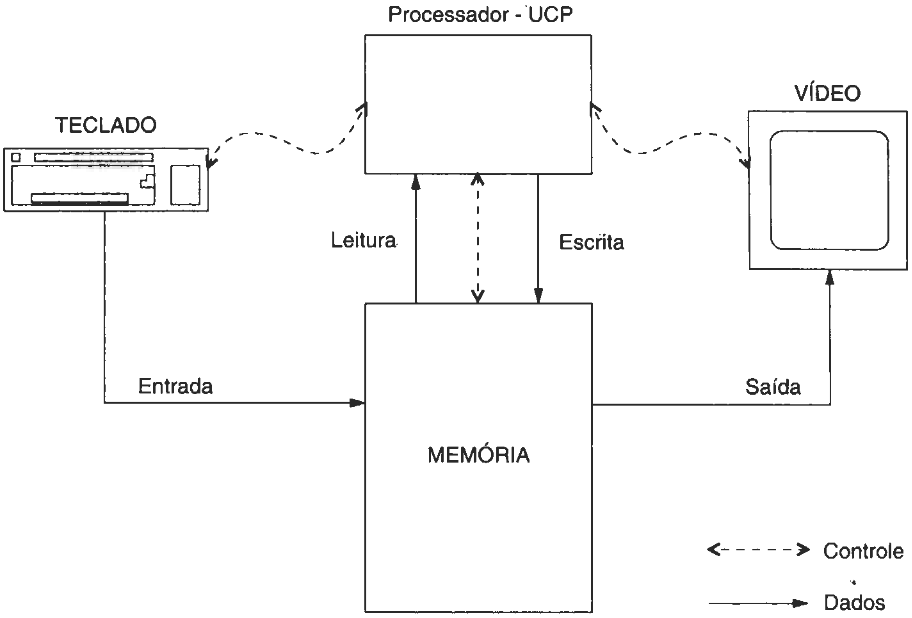


Figura 2.2 Componentes de um sistema de computação.

apresentados em um vídeo, como o que está mostrado na Fig. 2.2, ou impressos em um outro equipamento muito popular em computação, conhecido como Impressora, ou em qualquer outro dispositivo de SAÍDA (ver Cap. 10).

A quantidade de elementos eletrônicos individuais, a complexidade do processo de fabricação e do funcionamento, a capacidade e a velocidade de operação de cada um dos componentes de um computador (dispositivos de entrada, UCP, memória e dispositivos de saída) anteriormente descritas podem variar substancialmente de um sistema de computação para outro. Por exemplo, o antigo e já obsoleto processador Intel 8088, utilizado nos sistemas IBM-PC originais, era fabricado em uma pastilha com poucos milhares de transistores, tinha a capacidade de realizar 2,5 milhões de instruções por segundo, ou 2,5 MIPS, e podia trabalhar com uma memória capaz de armazenar 1.048.576 bytes (1 MB), enquanto o supercomputador CRAY Y-MP tem a capacidade de realizar de dois a quatro bilhões de operações aritméticas por segundo e pode trabalhar com uma memória de até 128 milhões de palavras (não se pode comparar a quantidade de transistores, pois ele não é implementado em um único *chip* — pastilha); o processador Pentium original (atualmente há no mercado vários modelos de Pentium, sendo o mais moderno o Pentium 4. O Pentium original pode ser chamado de Pentium I), lançado pela Intel em 1993, possuía cerca de 3,1 milhões de transistores na pastilha e podia endereçar até quatro bilhões de células de memória, enquanto o Pentium II possui 7,5 milhões de transistores e endereça até 64 gigabytes (células com capacidade de armazenar 1 byte). No entanto, as funções exercidas por esses componentes são sempre bastante semelhantes (todos os processadores — Intel 8088, CRAY Y-MP e Pentium executam o programa da Fig. 2.1(b), utilizando de forma semelhante a memória, a entrada, a saída e o processador).

A *Unidade Central de Processamento* é o componente vital do sistema, porque, além de efetivamente realizar as ações finais (as operações matemáticas com os dados), interpreta o tipo e o modo de execução de uma instrução, bem como controla quando e o que deve ser realizado pelos demais componentes, emitindo para isso sinais apropriados de controle (linhas tracejadas da Fig. 2.2).

Os programas e dados são armazenados na *Memória* para execução imediata (memória principal e memória cache) ou para execução ou uso posterior (memória secundária), conforme será discutido no Cap. 5.

Os dispositivos de *Entrada ou Saída* servem basicamente para permitir que o sistema de computação se comunique com o mundo exterior, realizando ainda, além da interligação, a conversão das linguagens do sistema para a linguagem do meio exterior (caracteres de nossas linguagens) e vice-versa. Os seres humanos entendem símbolos como A, b, ., (, + etc. e o computador entende sinais elétricos que podem assumir um valor de tensão (+3 volts) para representar o valor 1 ou um outro valor (0 volt) para representar o valor 0. O teclado (dispositivo de ENTRADA) interliga o usuário (mundo exterior) e o computador, permitindo a comunicação entre ambos através do uso das suas teclas. Ao ser pressionada a tecla correspondente ao caractere A, por exemplo, os circuitos eletrônicos existentes no teclado "convertem" (ver Cap. 10) a pressão mecânica em um grupo de sinais elétricos, alguns com voltagem alta (bit 1) e outros com voltagem baixa (bit 0), que corresponde, para o computador, ao caractere A.

Os dispositivos de SAÍDA operam de modo semelhante, porém em sentido inverso, isto é, do computador para o exterior, convertendo os sinais elétricos internos (e que, em grupos, podem representar um caractere inteligível pelo ser humano) em símbolos conhecidos pelos humanos, como os caracteres C, e, h, *, >, + etc.

No Cap. 10 serão mostradas as características básicas de alguns dos dispositivos de saída mais conhecidos, como o vídeo, a impressora, os discos magnéticos e o mouse.

Em resumo, os sistemas atuais, embora mais potentes, possuem os mesmos componentes básicos e realizam suas funções essenciais orientadas pelos mesmos conceitos fundamentais expostos no relatório apresentado por John von Neumann [NEUM 45], relativo à arquitetura do seu sistema EDVAC e do IAS (ver item 1.2.4.1), quais sejam:

- dados e instruções são armazenados em uma memória do tipo que escreve e recupera (leitura) (ver Cap. 5);
- o conteúdo da memória é endereçado conforme a sua posição, independentemente do tipo da informação nele contido (ver Cap. 5); e
- a execução das instruções ocorre de forma seqüencial (a não ser que uma instrução específica mude momentaneamente a seqüência), uma em seguida à outra — ver item 6.4.

2.2 REPRESENTAÇÃO DAS INFORMAÇÕES

2.2.1 O Bit, o Caractere, o Byte e a Palavra

Toda informação introduzida em um computador (sejam dados que serão processados ou instruções de um programa) precisa ser entendida pela máquina, para que possa corretamente interpretá-la e processá-la.

No texto deste livro, as informações apresentadas em forma de caracteres são entendidas pelo leitor porque ele conhece o formato e o significado dos símbolos que representam os caracteres alfabéticos, os numéricos (algarismos) e os sinais de pontuação ou matemáticos (+, − , ×, /, >, <, = etc.).

O computador, sendo um equipamento eletrônico, armazena e movimenta as informações internamente sob forma eletrônica; esta pode ser um valor de voltagem ou de corrente (sabemos também que na memória secundária as informações são armazenadas sob forma magnética ou ótica).

Para que esta máquina pudesse representar eletricamente todos os símbolos utilizados na linguagem humana, seriam necessários mais de 100 diferentes valores de voltagem (ou de corrente). Tal máquina certamente seria difícil de ser construída para fins comerciais e, possivelmente, teria confiabilidade muito baixa (uma das grandes desvantagens do primeiro computador eletrônico construído, o ENIAC, foi justamente o fato de ser uma máquina decimal, o que foi imediatamente corrigido a partir da máquina seguinte, o IAS, que já era um computador binário [ver item 1.2.4.1].

No caso do IAS, optou-se por uma máquina binária já que von Neumann e sua equipe consideraram que seria muito mais simples e confiável projetar um circuito capaz de gerar e manipular o menor número possível de valores distintos, isto é, capaz de entender apenas dois valores diferentes: 0 e 1.

Além disso, com uma máquina binária, torna-se mais simples o emprego da lógica booleana (do SIM/NÃO, ABERTO/FECHADO, ACIMA/ABAIXO, LIGADO /DESLIGADO etc.).

Dessa forma, os computadores digitais (que trabalham com valores discretos) são totalmente binários. Toda informação introduzida em um computador é convertida para a forma binária, através do emprego de um código qualquer de armazenamento, como veremos mais adiante.

As linguagens utilizadas pelos humanos, como o português, possuem uma estrutura de informação criada para permitir a construção dos elementos necessários à comunicação entre pessoas, seja no formato falado seja no escrito. Assim é que nos comunicamos uns com os outros através de trechos do conjunto de elementos disponíveis na nossa linguagem, como os caracteres e as palavras, unindo-os de acordo com as regras de construção estabelecidas (léxica e de sintaxe).

O menor elemento disponível de uma linguagem humana é o caractere (em português, possuímos 23 caracteres alfabéticos, como o "a", o "d", o "t", 10 caracteres numéricos, como os algarismos "0", "1" e "9", além dos sinais de pontuação e de operações aritméticas, enquanto, na língua inglesa, há 26 caracteres alfabéticos, os nossos 23 mais o K, o Y e o W).

A menor unidade de informação armazenável em um computador é o algarismo binário ou dígito binário, conhecido como *bit* (contração das palavras inglesas *bi*nary digi*t*). O *bit* pode ter, então, somente dois valores: **0** e **1** (ver Fig. 7.1).

No entanto, um caractere isoladamente posto praticamente nada significa para nosso sentido de comunicação, razão por que criaram-se as palavras que são conjuntos de caracteres formando um sentido de informação útil, como "mesa", "pessoa", "carro" e uma enorme quantidade de outras palavras conforme podemos observar ao folhear um dicionário da língua portuguesa ou de outra língua qualquer.

Da mesma forma que na nossa linguagem a menor unidade de informação (o caractere) pouco ou nada significa como informação útil, em computação, com possibilidades tão limitadas, o bit pouco pode representar isoladamente; por essa razão, as informações manipuladas por um computador são codificadas em grupos ordenados de bits, de modo a terem um significado útil.

O menor grupo ordenado de bits que pode representar uma informação em computadores é o caractere da linguagem dos humanos, justamente a menor unidade de informação das nossas linguagens (como se pode verificar, em computação há um nível ainda mais baixo de representação de informação, que é o bit).

Qualquer caractere a ser armazenado em um sistema de computação é convertido em um conjunto de bits previamente definido para o referido sistema (chama-se *código de representação de caracteres).*

Cada sistema poderá definir como (quantos bits e como se organizam) cada conjunto de bits irá representar um determinado caractere. Poderão, por exemplo, ser cinco bits por caractere (nesse caso, serão codificados 32 símbolos diferentes), seis bits por caractere (codificando 64 símbolos diferentes), sete bits, oito bits, e assim por diante. No Cap. 7, serão apresentados alguns dos principais códigos de representação de caracteres (ver também Apêndice B).

A primeira definição formal atribuída a um grupo ordenado de bits, para efeito de manipulação interna mais eficiente, foi instituída pela IBM e é, atualmente, utilizada por praticamente todos os fabricantes de computadores. Trata-se do *byte,* definido como um grupo ordenado de oito bits, tratados de forma individual, como unidade de armazenamento e transferência.

O byte foi definido para servir de elemento de referência para a construção e funcionamento dos dispositivos de armazenamento e também como referência para os processos de transferência de dados entre periféricos e UCP/MP. As impressoras continuam recebendo dados byte a byte, como também é costume no mercado construírem memórias cujo acesso, armazenamento e recuperação de informações são efetuados byte a byte (ou caractere a caractere). Por essa razão, em anúncios de computadores, menciona-se que ele possui "512 Kbytes de memória" ou "32M caracteres de memória" ou ainda "2 Gbytes de memória", por exemplo. Na realidade, em face desse costume, quase sempre o termo byte é omitido por já subentender esse valor.

Como os principais códigos de representação de caracteres utilizam grupos de oito bits por caractere, os conceitos de *byte* e *caractere* tornam-se semelhantes, e as palavras, quase sinônimas. O termo *caractere* é mais empregado para fins comerciais (propaganda, apresentações a pessoas não familiarizadas com o jargão de computação), enquanto o termo *byte* é empregado mais na linguagem técnica dos profissionais da área.

No entanto, é bom prestar atenção ao fato de que, embora um caractere represente pouca informação com sentido (por exemplo, a letra "a" não fornece nenhuma informação se estiver isolada, como a maioria das letras de nosso alfabeto, que nada representam se estiverem isoladas em um texto, como o "l", o "v"), ainda assim trata-se de um elemento bem definido de informação. Já o byte, que pode representar um caractere internamente no computador, não tem a finalidade de representar qualquer tipo de informação, sendo tão-somente uma unidade de armazenamento e transferência.

Esse fato é particularmente verdadeiro quando se trata de valores numéricos, que usualmente são representados em um sistema de computação por uma quantidade de bits bem maior do que a de um byte (8 bits). Assim é que, em um determinado sistema de computação, os números podem ser representados com conjuntos de 32 bits, o que compreende 4 bytes de dados para cada número. Nesse caso, um byte nada representa, pois é apenas parte do valor do número (é um caso semelhante ao de nosso sistema numérico, o decimal, onde podemos ter que escrever em um papel o número 1539734, que possui 7 algarismos. No caso desse número, o algarismo 7 isoladamente nada representa, nem qualquer um dos demais algarismos).

Voltando ao exemplo anterior sobre a utilização do termo byte em citações sobre a capacidade de memória de computadores, verificamos a inclusão dos caracteres K, M e G. Tais caracteres são letras indicativas de um valor numérico fixo, utilizado para reduzir a quantidade de algarismos representativos de um número. Nas grandezas métricas, usa-se o K para representar mil vezes.

Como os computadores são máquinas binárias, todas as indicações numéricas referem-se a potências de 2 e não a potências de 10 como no sistema métrico e, por essa razão, o K representa 1024 unidades (décima potência de 2 ou $2^{10} = 1024$), o M (abreviatura do termo mega) representa 1.048.576 unidades (valor igual a 1024 × 1024 ou $2^{10} \times 2^{10} = 2^{20}$) e o *giga,* representado pelo caractere G, indica um valor igual a 1024 mega ou 1.048.576K ou $2^{30} = 2^{10} \times 2^{10} \times 2^{10} = 1.073.741.824$ unidades.

Em conseqüência, no exemplo anterior, o valor 512 Kbytes (dizemos quinhentos e doze "ka" bytes) corresponde a um valor de 512 × 1024 = 524.288 bytes, enquanto 32M caracteres ("trinta e dois mega caracteres") corresponde a 32 × 1024 × 1024 = 33.554.432 caracteres e 2 Gbytes ("um giga bytes") corresponde a 2 × 1024 × 1024 × 1024 = 2.147.483.648 bytes.

Com o progressivo aumento da capacidade dos dispositivos de armazenamento dos computadores, criaram-se mais dois elementos para abreviar valores mais elevados: trata-se do termo *tera,* para representar um valor igual a 2^{40}, ou 1024G, e do *peta* para representar 2^{50}, ou 1024 teras (e certamente, em um futuro próximo, estaremos utilizando outras abreviaturas mais poderosas).

Dessa forma, os valores utilizados em computação para indicar capacidade de memória são normalmente compostos de um número (sempre entre 0 e 999) e uma das abreviaturas citadas. A Tabela 2.1 mostra alguns exemplos de grandezas utilizadas em computação.

Conforme observamos anteriormente, a estrutura das linguagens dos humanos inicia pelo caractere e segue organizando grupos de caracteres para formar, aí sim, uma unidade útil de informação, as palavras.

Também em computação (e penso que pelo mesmo motivo), criou-se o conceito da palavra, embora nesse caso ele tenha pequenas diferenças em relação às palavras das nossas linguagens. Assim, além do bit e do byte, temos o conceito relacionado com o armazenamento e a transferência de informações entre MP e UCP, porém mais especialmente relacionado ao processamento de dados pela UCP, denominado *palavra.*

Inicialmente, podemos definir a *palavra* como um conjunto de bits que representa uma informação útil para os computadores. Desse modo, uma palavra estaria associada ao tipo de interação entre MP e UCP, que é individual, informação por informação. Ou seja, a UCP processa instrução por instrução (cada uma estaria associada a uma palavra), armazena ou recupera número a número (cada um estaria associado a uma palavra), e assim por diante. Na prática, há diferenças em relação a esta idéia.

Tabela 2.1 Exemplos de Valores Utilizados para Indicar Grandezas em Computação

1K = 1024
1M = 1024K = 1024 × 1024 = 1.048.576
1G = 1024M = 1.048.576K = 1.048.576 × 1024 = 1.073.741.824
256K = 256 × 1024 = 262.124
64M = 64 × 1024 × 1024 = 65.536K = 65.536 × 1024 = 67.108.864
16G = 16 × 1024M = 16.384 × 1024 = 16.777.216K = 16.777.216 × 1024 = 17.179.869.184

Tabela 2.2 Estrutura de Informações nas Linguagens dos Humanos e nos Computadores

Computadores	Linguagens dos humanos
Bit	Caractere
Byte e caractere	Palavra
Palavra	Frases
Registro	Textos
Arquivo	Livros
Banco de dados	

A palavra nos computadores é um valor fixo e constante para um dado processador (32 bits, como nos Pentium e Motorola, ou 64 bits, como no mais novo processador a ser lançado pela Intel e os Alpha), diferentemente das linguagens dos humanos onde as palavras têm quantidades variáveis de caracteres (mesa possui 4 caracteres, enquanto automóvel possui 9 caracteres).

O conceito de palavra não é rigorosamente igual para todos os fabricantes. Alguns estabelecem o tamanho dos registradores internos da UCP igual ao da palavra, enquanto outros usam este conceito de palavra de modo mais abrangente. A Intel, AMD e Motorola, para seus microprocessadores, seguem os mesmos conceitos antigos da IBM.

No que se refere à unidade de armazenamento, considera-se mais importante a quantidade de bits recuperada em um acesso, em geral de tamanho igual ao de um byte. Esse valor de bits é pequeno demais para representar um número ou uma instrução de máquina e, por isso, não pode ser aceitável para o tamanho de uma palavra.

De modo geral, usam-se dois valores diferentes: um relacionado à *unidade de armazenamento — o byte* (oito bits é o valor mais comum) e outro para indicar a *unidade de transferência e processamento — a palavra* (que, na quase totalidade de computadores, possui um número de bits múltiplo de 1 byte — 16 ou 32 bits é o valor mais comum). Em geral, a UCP processa valores representados por uma quantidade de bits igual à da palavra, indicando assim a capacidade de processamento do sistema.

No item 6.2.1.3 pode-se verificar de modo mais claro a utilidade do conceito da palavra como unidade de processamento e até mesmo como unidade de transferência de dados internamente e não como unidade de armazenamento.

Nos próximos capítulos, os conceitos de palavra, de unidades de armazenamento e de transferência e o emprego do byte serão detalhadamente apresentados e exemplificados. A Tabela 2.2 apresenta um resumo dos conceitos já emitidos, ampliando a estrutura de informação nos computadores, mesmo com conceitos ainda não explicados, apenas para efetuar o devido registro e chamar a atenção do leitor.

2.2.2 Conceito de Arquivos e Registros

Todo processamento em um computador consiste, como já mencionado, na manipulação de dados segundo um conjunto de instruções que, globalmente, chamamos de *programa.*

Para que seja possível individualizar grupos diferentes de informações (o conjunto de dados de um programa constitui um grupo diferente do conjunto de dados de outro programa, por exemplo), os sistemas operacionais (programas que controlam o armazenamento e recuperação dessas informações para entrada, saída ou guarda em memória secundária) estruturam esses grupos de dados sob uma forma denominada *arquivo.*

Um arquivo de informações (ou dados) é um conjunto formado por dados (ou informações) de um mesmo tipo ou para uma mesma aplicação. Por exemplo, podemos ter um arquivo de alunos de uma turma (contendo informações sobre cada aluno individualmente) ou um arquivo contendo as instruções de um programa.

Cada arquivo é constituído por itens individuais de informação (cada aluno, no nosso exemplo) chamados *registros.*

Assim, um arquivo de uma turma de 60 alunos possui um total de 60 registros; um arquivo com informações sobre mil empregados de uma organização possui mil registros, e assim por diante.

Um programa é também um arquivo (embora constituído de um único registro, visto que as instruções não são consideradas como registros individuais).

Para entendermos melhor o conceito de armazenamento e recuperação de informações sob a forma de arquivos, podemos fazer analogia com um sistema semelhante, porém manual.

Suponhamos a existência de uma empresa com 500 empregados, que manipula um estoque de material de consumo com cerca de 10 mil itens e que, por incrível que possa parecer, ainda não possua um sistema de computação eletrônico.

Na gerência de pessoal, as informações sobre os funcionários da empresa estão organizadas da seguinte forma:

- as informações sobre cada funcionário são colocadas em um formulário apropriado, estruturado com campos separados para cada um dos itens de informação, tais como número de matrícula, nome, endereço, departamento, salário;
- o formulário é guardado ("armazenado") em uma pasta (uma para cada funcionário), identificada externamente pelo número de matrícula do funcionário;
- as 500 pastas são guardadas em um armário de aço com gavetas (arquivo), sendo organizadas em ordem crescente de número de matrícula (é a chave de acesso à pasta desejada).

Já a gerência de material possui controle dos itens do estoque de material de forma semelhante: há uma ficha para cada item de estoque, contendo as informações necessárias sobre cada um; as fichas são armazenadas em pequenas caixas metálicas, chamadas *arquivos portáteis.*

No exemplo descrito, o armário de aço e a caixa metálica constituem arquivos, com função semelhante aos arquivos de dados em sistemas de computação eletrônicos; cada pasta ou ficha constitui um registro, respectivamente, do funcionário ou do material de estoque. As informações de um funcionário são especificadas em campos separados (nome é um campo, endereço é outro campo, e assim por diante).

A estrutura de armazenamento e recuperação de informações na memória secundária de um sistema de computação é concebida segundo o conceito de arquivos e registros. Isso porque, na memória secundária, o sistema operacional pode guardar informações em grupos para obter maior eficiência na transferência com a memória principal.

O processo é diferente da estrutura da memória principal, onde a preocupação é com itens individuais de informação (uma instrução, um número, uma letra etc.).

2.3 CLASSIFICAÇÃO DE SISTEMAS DE COMPUTAÇÃO

Atualmente, quando se deseja adquirir um sistema de computação para realização de alguma atividade, há milhares de opções, as quais podem ser classificadas de modo genérico (esta classificação não é positiva nem consensual no mercado, na indústria ou no meio acadêmico, porém é um razoável auxílio para quem vai adquirir um sistema, a fim de definir suas necessidades) em:

- microcomputadores (desktops, laptops, notebooks, palmtops);
- estações de trabalho (workstations);
- minicomputadores;
- computadores de grande porte (mainframes); e
- supercomputadores.

Atualmente, os microcomputadores dominam o mercado, sendo utilizados em larga escala nas empresas, órgãos governamentais e como computadores pessoais. Como foi exposto no Cap. 1, eles surgiram por volta de 1974, através do desenvolvimento dos microprocessadores (todos os componentes de uma UCP em uma única pastilha) e o nome foi dado justamente devido ao tamanho e à capacidade de processamento, ambos pequenos em relação aos sistemas que já existiam no mercado.

Microcomputadores também se referiam ao tipo de usuário, no caso uma única pessoa, e, por isso, eram conhecidos como computadores pessoais (Personal Computers — PC). Atualmente, o nome microcomputador existe porque o mercado e os usuários se acostumaram, porque o computador só tem de micro o tamanho dos componentes e o seu próprio, já que sua capacidade de processamento e de armazenamento é tão grande quanto o de qualquer outro tipo. Essas máquinas se desenvolveram tecnologicamente de forma tão vertiginosa, nos últimos anos, que se tornaram as máquinas mais utilizadas na área comercial, a ponto de balançar empresas do tipo da IBM e da DEC, obrigando-as à reorganização interna e permitiram o surgimento de empresas voltadas somente para a fabricação de microcomputadores e que hoje são gigantes do mercado, como a Compaq, a Microsoft, a Dell, a Sun e outras.

Com o tempo, foram surgindo no mercado várias categorias de microcomputadores, cuja classificação está muito relacionada ao tamanho físico do equipamento e ao seu grau de portabilidade. Os micros (é mais comum e popular denominá-los assim do que pela sua forma completa — microcomputador) podem ser do tipo *de mesa* (*desktop*), em geral constituídos de três unidades fisicamente separadas, posicionadas em cima de uma mesa qualquer: a *unidade de processamento,* que possui internamente a UCP, os acionadores de disquetes, o disco de maior capacidade, a chave de alimentação e a memória principal; a *unidade de vídeo* e o *teclado.* Uma variação deste tipo consiste na substituição da unidade de processamento de mesa por uma outra chamada *torre,* por se tratar de um equipamento construído em forma vertical, estreita como uma torre, e com maior disponibilidade para instalação de dispositivos de entrada e saída. Há pouco tempo, com o aperfeiçoamento da tecnologia de microeletrônica, começaram a ser produzidos microcomputadores ainda menores e portáteis, que podem ser energizados por corrente elétrica ou por uma bateria embutida no equipamento. São chamados de diversos nomes, dependendo do seu tamanho: os maiores foram denominados *Laptops,* em seguida apareceram os *Notebooks,* de tamanho menor, em formato parecido com um livro. Há também os *Subnotebooks,* ainda menores, e os *Palmtops,* porque quase são do tamanho da palma de nossas mãos.

A título de exemplo para o leitor iniciante, as Figs. 2.3 a 2.6 mostram modelos de alguns dos equipamentos classificados anteriormente.

No Cap. 1 já mencionamos que o primeiro microcomputador, lançado comercialmente em dezembro de 1974, foi um modelo para montagem pelo usuário, chamado ALTAIR, que usava o microprocessador Intel 8080. Posteriormente, vários outros modelos surgiram, utilizando aquele microprocessador ou outros já existentes na época, como o Motorola 6800, o Z-80 e o National 6502. Em 1981 a IBM, que se dedicava a ganhar dinheiro com a comercialização de hardware e software para grandes sistemas de computação, agitou o mercado de microcomputação com o lançamento de seu PC — Personal Computer (computador pessoal), baseado em um microprocessador Intel 8088 (ver Cap. 1). A ele sucederam-se dezenas e até centenas de imitações, a mais importante delas sendo de uma companhia denominada Compaq. No que se refere aos mi-

Figura 2.3 Um notebook.

Figura 2.4 Um microcomputador do tipo *desktop*.

crocomputadores portáteis, a Toshiba liderou o mercado durante um bom tempo, sendo agora ameaçada por vários fabricantes, entre os quais IBM, NEC, Compaq, Dell e Gateway.

Uma *estação de trabalho* é essencialmente um microcomputador projetado para realizar tarefas pesadas, em geral na área científica ou industrial, como complexas computações matemáticas, projetos com auxílio de computação (CAD — Computer Aided Design), fabricação com auxílio de computação (CAM — Computer Aided Manufacturing) e a composição, manipulação e apresentação de gráficos e imagens de altíssima re-

Figura 2.5 Um palmtop.

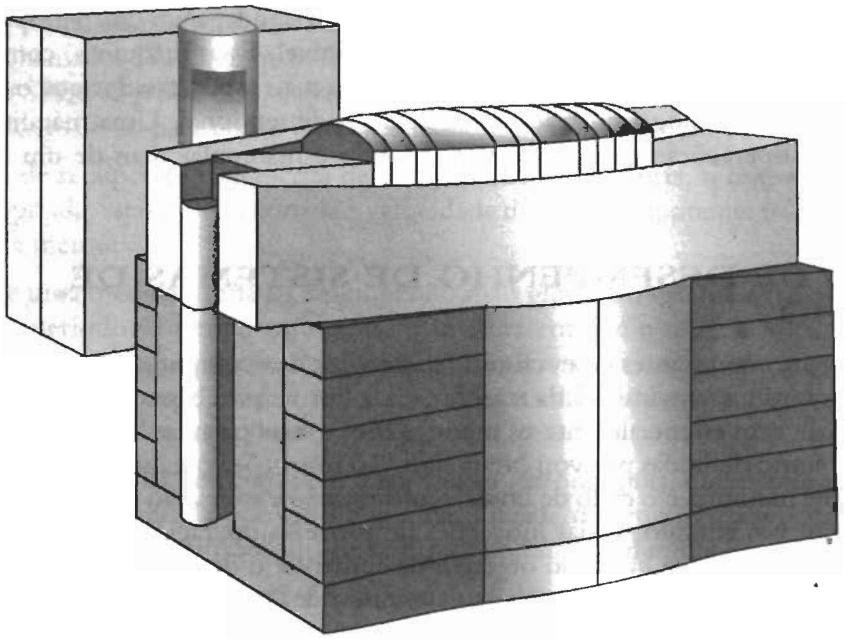


Figura 2.6 Um supercomputador, o Cray T-90.

solução. Essas tarefas requerem mais velocidade de processamento, mais capacidade de memória (em tamanho e velocidade de transferência de informações) e dispositivo de vídeo de mais alta qualidade do que as características usuais dos microcomputadores.

Especialmente no que se refere à velocidade do processador e à capacidade de memória, a potência de uma estação de trabalho é semelhante à de um minicomputador. Entretanto, as estações de trabalho são dirigidas ao usuário, ao contrário dos minicomputadores. O sistema DECstation 5000/33 ou IBM RS/6000 são exemplos de estações de trabalho, como também as fabricadas pela Sun Microsystems (Sparc), todos utilizando uma arquitetura diversa das dos microcomputadores do tipo PC, denominada de RISC — Reduced Instruction Set Computer (computador com conjunto reduzido de instruções), cujas características serão abordadas no Cap. 11.

Minicomputadores, por sua vez, são máquinas projetadas para atender simultaneamente à demanda por execução de programas de vários usuários, embora a quantidade de usuários e de programas não seja tão grande quanto se pode encontrar em computadores de grande porte.

A capacidade de suportar múltiplos usuários e programas requer, além de velocidade de processamento e capacidade/velocidade de memória, uma extensa potencialidade para manipular diversos dispositivos de entrada e saída. Os sistemas VAX-11/780 da DEC são exemplos típicos de minicomputadores, bem como os sistemas AS/400 da IBM.

Além disso, os minicomputadores requerem uma sofisticação maior dos programas de controle (sistemas operacionais), que somente agora está se tornando necessária nos microcomputadores. Tais sistemas praticamente não são mais fabricados.

Já os *computadores de grande porte* são sistemas projetados para manusear considerável volume de dados e executar simultaneamente programas de uma grande quantidade de usuários. Essas máquinas podem interagir com centenas de usuários em um dado instante, como, por exemplo, um sistema de reserva de passagens aéreas, onde há necessidade de armazenamento em grande escala (todos os dados de vôos e das reservas realizadas), bem como uma contínua solicitação de processamento por parte dos incontáveis terminais conectados diretamente ao sistema, aos quais o computador tem que atender e responder em poucos segundos. Os sistemas IBM 3090 e Control Data CDC 6600 são típicos exemplos de computadores de grande porte, também atualmente pouco usados devido a sua progressiva substituição por microcomputadores potentes ou máquinas RISC de maior capacidade e velocidade de processamento, ou ainda por redes de microcomputadores.

Finalmente, um *supercomputador* é projetado primariamente para atender a um único propósito: realizar grandes quantidades de cálculos matemáticos o mais rapidamente possível. Essas máquinas, como o sistema CRAY Y-MP e IBM 9021, podem realizar aplicações que demandam mais o processador que os demais componentes, tais como: previsão de tempo, simulação, modelagem tridimensional. Uma máquina dessas é capaz de realizar dois bilhões de operações matemáticas por segundo e manipular mais de um bilhão de células de memória.

2.4 MEDIDAS DE DESEMPENHO DE SISTEMAS DE COMPUTAÇÃO

Um dos aspectos mais interessantes da evolução tecnológica dos computadores é que, embora esta tenha sido, sem dúvida, extraordinariamente rápida e acentuada, os princípios básicos estabelecidos nos primórdios da computação permanecem essencialmente os mesmos. Em outras palavras, as características de arquitetura do computador IAS, como definidas por von Neumann, são instruções organizadas em um programa, previamente armazenadas na memória e o ciclo de busca, interpretação e execução de cada instrução, se mantendo de modo geral intactas. No entanto, várias inovações no processo operacional do ciclo das instruções vêm surgindo, ano após ano, sempre na busca do objetivo de aumentar o desempenho dos sistemas de computação. O desempenho aumenta dramaticamente, mas a essência dos princípios fundamentais de concepção do processo de computação tem permanecido, pelo menos até o momento atual.

Na busca do aumento de desempenho, verifica-se que a medida geral deste desempenho depende fundamentalmente da capacidade e velocidade de seus diferentes componentes, da velocidade com que estes componentes se comunicam entre si e do grau de compatibilidade que possa existir entre eles (por exemplo, se a velocidade da UCP de um sistema é muito maior que a da memória, então este sistema tem um desempenho inferior ao de um outro em que a UCP e a memória têm velocidades mais próximas).

Há muito tempo que se sabe que uma corrente constituída de vários elos interligados é tão mais forte quanto o mais fraco deles, assim como um sistema (constituído de vários componentes) é tão mais produtivo e eficaz quanto o menos produtivo e eficaz de seus componentes. Por exemplo, uma ligação telefônica de um aparelho localizado no Rio de Janeiro para outro localizado em Londres, Inglaterra, se realiza através da conexão de vários circuitos (como os elos de uma corrente), iniciando pela linha física que conecta o aparelho à sua central local no Rio de Janeiro, desta passando provavelmente por uma ou mais centrais até a da operadora internacional (ainda no Rio de Janeiro). Daí se estabelece uma conexão por satélite com a central da operadora internacional na Inglaterra e os circuitos vão se estabelecendo de modo similar na Inglaterra, em Londres, e até o aparelho de destino. Todos esses circuitos menos um podem ser da melhor qualidade, devido à excelente tecnologia de sua construção, o mais imunes a ruídos etc. Um deles, no entanto, pode ter sofrido algum tipo de problema e, neste caso, o sinal passando por ele sofre grande interferência de ruídos, sem proteção adequada para evitá-lo. As duas pessoas estarão falando em meio a ruídos e, naturalmente, reclamando da qualidade e do desempenho do circuito completo.

Assim também acontece com os computadores, constituídos de diversos componentes, como processador, memórias, barramentos, dispositivos periféricos e outros. Para aumentar o desempenho desses sistemas é necessário que todos os seus componentes tenham qualidade adequada para que um deles não comprometa o esforço dos demais.

Considerando a existência de tantos fatores que influenciam o desempenho de um sistema de computação, desenvolveram-se diversos meios de medir seu desempenho.

O desempenho dos processadores, em geral, é medido em termos da sua velocidade de trabalho. Como seu trabalho é executar instruções, criou-se a unidade (sempre questionada por alguns) chamada MIPS (milhões de instruções por segundo) e também a unidade MFLOPS (milhões de operações de ponto flutuante por segundo), que é uma medida típica de estações de trabalho e de supercomputadores, pois estes costumam trabalhar mais com cálculos matemáticos.

Para tentar equalizar e padronizar as medidas de desempenho de processadores de diferentes fabricantes e com características diferentes, foram desenvolvidos programas de teste e medida denominados SPEC (System Performance Evaluation Cooperative). Este processo teve origem com a criação, em 1989, de um grupo de trabalho, organizado por algumas empresas (Hewlett-Packard — HP, Sun, Mips e outras) e cujos programas em conjunto foram chamados SPEC, os quais foram sendo aperfeiçoados ao longo do tempo, redundando em novas versões, como SPEC92 e SPEC95. O conjunto compreende 16 programas de teste e medida, sendo 8 para cálculos com inteiros e 10 para ponto flutuante, denominando-se, assim, SPECint95 e SPECfp95.

Já quando se trata de recuperação ou escrita de informações na memória, o *tempo de acesso* é uma unidade de medida mais apropriada, estando relacionada à velocidade de cada componente e à do canal de interligação entre os dois (UCP e memória).

Tempo de resposta é uma medida ligada ao desempenho mais global do sistema e não de um ou outro componente. Trata-se do período de tempo gasto entre o instante em que o usuário iniciou uma solicitação ou interrogação e o instante em que o sistema apresentou ao usuário a sua resposta ou atendeu à sua solicitação. Por exemplo, o intervalo de tempo entre a solicitação de um saldo de conta em um terminal bancário e a apresentação da resposta no vídeo (o saldo da conta).

Uma outra unidade de medida de desempenho é a **vazão** (throughput), que define a quantidade de ações ou transações que pode ser realizada por um sistema na unidade de tempo. Por exemplo, a quantidade de atualizações que pode ser feita em um sistema de controle do estoque de uma empresa.

Quando estamos nos referindo à velocidade com que um determinado dispositivo de entrada ou de saída transfere ou recebe dados da UCP, utilizamos uma unidade que mede a taxa de transferência que o canal de ligação (ver barramento no item 6.5) pode suportar, isto é, a quantidade de bits por segundo que pode trafegar pelo referido canal.

EXERCÍCIOS

1) Explique o que você entende por memória. Cite dois exemplos de memórias na vida prática (evite usar exemplo de memória de computador).

2) Descreva as funções de uma Unidade Central de Processamento.

3) Faça o mesmo para a memória de um computador.

4) Para que servem os dispositivos de entrada e de saída de um computador? Cite alguns exemplos.

5) Imagine uma empresa qualquer. Cite exemplos de arquivos e registros a serem criados para o armazenamento das informações que circulam na tal empresa.

6) Conceitue o bit, o byte e a palavra.

7) Indique o valor de x nas seguintes expressões:

a) 65.536 = xK
b) 12.288K = xM
c) 19.922.944 = xM
d) 8 Gbytes = x bytes
e) 64 Kbytes = x bits
f) 262.144 bits = xK bits
g) 16.700.160 palavras = x palavras (usando a menor unidade possível)
h) 128 Gbits = x bits
i) 512 K células = x células
j) 256 Kbytes = x bits

8) O que é vazão em um sistema de computação? E tempo de resposta? Em que circunstâncias são utilizadas estas informações?

9) Em que um supercomputador difere de um computador de grande porte?

10) Quais são as principais características que definem um microcomputador?

11) Qual é a diferença entre linguagem de alto nível e linguagem de máquina?

12) Cite os conceitos fundamentais de um computador que se inspire na arquitetura proposta por John von Neumann.

13) Cite exemplos de UCP comerciais.

3

Conversão de Bases e Aritmética Computacional

3.1 NOTAÇÃO POSICIONAL — BASE DECIMAL

Desde os primórdios da civilização o Homem vem adotando formas e métodos específicos para representar números, tornando possível, com eles, contar objetos e efetuar operações aritméticas (de soma, subtração etc.).

A forma mais empregada de representação numérica é a chamada *notação posicional*. Nela, os algarismos componentes de um número assumem valores diferentes, dependendo de sua posição relativa no número. O valor total do número é a soma dos valores relativos de cada algarismo. Desse modo, é a posição do algarismo ou dígito que determina seu valor.

A formação de números e as operações com eles efetuadas dependem, nos sistemas posicionais, da quantidade de algarismos diferentes disponíveis no referido sistema. Há muito tempo a cultura ocidental adotou um sistema de numeração que possui dez diferentes algarismos — 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 — e, por essa razão, foi chamado de *sistema decimal*. (Ver mais detalhes no Apêndice A — Sistemas de Numeração.)

A quantidade de algarismos disponíveis em um dado sistema de numeração é chamada de *base*; a base serve para contarmos grandezas maiores, indicando a noção de grupamento. O sistema de dez algarismos, mencionado anteriormente, tem base 10; um outro sistema que possua apenas dois algarismos diferentes (0 e 1) é de base 2, e assim por diante.

Vamos exemplificar o conceito de sistema posicional. Seja o número 1303, representado na base 10, escrito da seguinte forma:

$1\,3\,0\,3_{10}$

Em base decimal, por ser a mais usual, costuma-se dispensar o indicador da base, escrevendo-se apenas o número:

1303

Neste exemplo, o número é composto de quatro algarismos:

1, 3, 0 e 3

e cada algarismo possui um valor correspondente à sua posição no número.

Assim, o primeiro 3 (algarismo mais à direita) representa 3 unidades. Neste caso, o valor absoluto do algarismo (que é 3) é igual ao seu valor relativo (que também é 3), por se tratar da 1.ª posição (posição mais à direita, que é a ordem das unidades). Considerando-se o produto três vezes a potência 0 da base 10 ou

$3 \times 10^0 = 3$

enquanto o segundo 3 vale três vezes a potência 2 da base 10 ou

$3 \times 10^2 = 300$

E o último à esquerda vale 1 vez a potência 3 da base 10 ou $1 \times 10^3 = 1000$.

O valor total do número seria então:

$1000 + 300 + 0 + 3 = 1303_{10}$

$1 \times 10^3 + 3 \times 10^2 + 0 \times 10^1 + 3 \times 10^0 = 1303_{10}$

Generalizando, num sistema qualquer de numeração posicional, um número N é expresso da seguinte forma:

$$N = (d_{n-1}\, d_{n-2}\, d_{n-3} \ldots d_1\, d_0)_b \qquad (1.1)$$

onde:

d indica cada algarismo do número;

n − 1, n − 2, 1, 0 (índice) indicam a posição de cada algarismo;

b indica a base de numeração;

n indica o número de dígitos inteiros.

O valor do número pode ser obtido do seguinte somatório:

$$N = d_{n-1} \times b^{n-1} + d_{n-2} \times b^{n-2} + \ldots + d_1 \times b^1 + d_0 \times b^0 \qquad (1.2)$$

Desse modo, na base 10, podemos representar um número:

$N = 3748$

onde:

$n = 4$ (quatro dígitos inteiros).

Utilizando a fórmula indicada na Eq. 1.1:

$d_{n-1} = 3$ ou $d_3 = 3$; $d_2 = 7$; $d_1 = 4$; $d_0 = 8$

ou obtendo seu valor de acordo com a fórmula mostrada em (1.2):

$$N = 3 \times 10^3 + 7 \times 10^2 + 4 \times 10^1 + 8 \times 10^0 =$$
$$= 3000 + 700 + 40 + 8 = 3748_{10}$$

Observação: Números fracionários são apresentados em detalhe no Apêndice A.

3.2 OUTRAS BASES DE NUMERAÇÃO

Vejamos, em seguida, como representar números em outra base de numeração.

Entre as bases diferentes de 10, consideremos apenas as bases 2 e potências de 2, visto que todo computador digital representa internamente as informações em algarismos binários, ou seja, trabalha em base 2. Como os números representados em base 2 são muito extensos (quanto menor a base de numeração, maior é a quantidade de algarismos necessários para indicar um dado valor) e, portanto, de difícil manipulação visual, costuma-se representar externamente os valores binários em outras bases de valor mais elevado. Isso permite maior compactação de algarismos e melhor visualização dos valores. Em geral, usam-se as bases octal ou hexadecimal, em vez da base decimal, por ser mais simples e rápido converter valores binários (base 2) para valores em bases múltiplas de 2.

Utilizando-se a notação posicional indicada na Eq. 1.1, representam-se números em qualquer base:

$(1011)_2$ — na base 2

$(342)_5$ — na base 5

$(257)_8$ — na base 8

No entanto, nas bases diferentes de 10, o valor relativo do algarismo (valor dependente de sua posição no número) é normalmente calculado usando-se os valores resultantes de operações aritméticas em base 10 e não na base do número (ver Apêndice A para maiores detalhes) e, portanto, o valor total do número na base usada.

Exemplo 3.1

- Seja o número na base 2: $(1011)_2$ (usou-se a descrição da Eq. 1.1).
- Se aplicássemos a Eq. 1.2, teríamos:

  $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 =$

  $= 8 + 0 + 2 + 1 = (11)_{10}$
- Este valor 11 está expresso na base 10 e não na base 2. Portanto, será $(11)_{10}$.

Exemplo 3.2

$(1043)_5 = 1 \times 5^3 + 0 \times 5^2 + 4 \times 5^1 + 3 \times 5^0 =$

$= 125 + 0 + 20 + 3 = (148)_{10}$

Sobre o assunto, podemos concluir:

a) O número máximo de algarismos diferentes de uma base é igual ao valor da base.

   Exemplo:
   - na base 10 temos 10 dígitos: de 0 a 9;
   - na base 2 temos apenas dois dígitos: 0 e 1;
   - na base 5 temos cinco dígitos: de 0 a 4.

b) O valor do algarismo mais à esquerda (mais significativo) de um número de **n** algarismos inteiros é obtido pela multiplicação de seu valor absoluto (algarismo d_{n-1}) pela base elevada à potência **(n − 1)**, ou seja, $(d_{n-1} \times b^{n-1})$.

c) O valor total do número é obtido somando-se **n** valores, cada um expressando o valor relativo de um dos **n** algarismos componentes do número.

Exemplo 3.3

a) 375_{10}

   $n = 3$ (3 algarismos)

   $3 \times 10^2 + 7 \times 10^1 + 5 \times 10^0 =$ (3 produtos)

   $= 300 + 70 + 5$ (3 valores a somar)

b) 11101_2 (5 algarismos)

   $\underline{1 \times 2^4} + \underline{1 \times 2^3} + \underline{1 \times 2^2} + \underline{0 \times 2^1} + \underline{1 \times 2^0}$ (5 produtos − 5 valores)

   1.º prod. 2.º prod. 3.º prod. 4.º prod. 5.º prod.

   $16 + 8 + 4 + 0 + 1 = 29_{10}$

A base do sistema binário é 2 e, conseqüentemente, qualquer número, quando representado nesse sistema, consiste exclusivamente em dígitos 0 e 1. O termo dígito binário é chamado *bit*, contração do termo inglês *bi*nary digi*t*.

Por exemplo, o número binário 11011 possui cinco dígitos, ou algarismos binários. Diz-se que o referido número é constituído de 5 bits.

Em bases de valor superior a 10, usam-se letras do alfabeto para a representação de algarismos maiores que 9. Uma dessas bases é especialmente importante em computação — trata-se da base 16 ou hexadecimal, por ser de valor potência de 2 (como a base 8).

Nessa base, os "algarismos" A, B, C, D, E e F representam, respectivamente, os valores (da base 10): 10, 11, 12, 13, 14 e 15.

Na base 16 (hexadecimal), dispomos de 16 algarismos (não números) diferentes:

0, 1, 2, 3, ..., 9, A, B, C, D, E e F

Um número nessa base é representado na forma da Eq. 1.1:

$(1A7B)_{16}$

O seu valor na base 10 será obtido usando-se a Eq. 1.2:

$1 \times 16^3 + 10 \times 16^2 + 7 \times 16^1 + 11 \times 16^0 = 4096 + 2560 + 112 + 11 = 6779_{10}$

Observemos que na Eq. 1.2 foram usados os valores 10 (para o algarismo A) e 11 (para o algarismo B) para multiplicar as potências de 16. Por isso, obtivemos o valor do número na base 10.

Em outras palavras, utilizamos valores e regras de aritmética da base 10 e, por isso, o resultado encontrado é um valor decimal. A Tabela 3.1 mostra a representação de números nas bases 2, 8, 10 e 16.

Pela tabela, podemos observar que os dígitos octais e hexadecimais correspondem a combinações de 3 (octais) e 4 (hexadecimais) bits (algarismos binários). Sendo a base desses sistemas de valor maior que a base 2 e tendo em vista essa particularidade na representação de números nas bases 8 e 16 em relação à base 2, verifica-se que é possível converter rapidamente números da base 2 para as bases 8 ou 16, ou vice-versa.

Por exemplo, o número (101111011101), na base 2, possui 12 algarismos (bits), mas pode ser representado com quatro algarismos octais ou com apenas três algarismos hexadecimais:

$(101111011101)_2 = (5735)_8$

porque 101 = 5; 111 = 7; 011 = 3 e 101 = 5

$(101111011101)_2 = (BDD)_{16}$

porque 1011 = B; 1101 = D; 1101 = D.

Tabela 3.1

Base 2	Base 8	Base 10	Base 16
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10
10001	21	17	11

3.3 CONVERSÃO DE BASES

Uma vez entendido como representar números em notação posicional, e como esta notação é aplicável em qualquer base inteira, podemos exercitar a conversão de números de uma base para outra.

Interessa-nos, principalmente, verificar o processo de conversão entre bases múltiplas de 2, e entre estas e a base 10, e vice-versa.

3.3.1 Conversão entre Bases Potência de 2

3.3.1.1 Entre as Bases 2 e 8

Como $8 = 2^3$, um número binário (base 2) pode ser facilmente convertido para o seu valor equivalente na base 8 (octal). Se o número binário for inteiro, basta dividi-lo, da direita para a esquerda, em grupos de 3 bits (o último grupo, à esquerda, não sendo múltiplo de 3, preenche-se com zeros à esquerda). Então, para cada grupo, acha-se o algarismo octal equivalente, conforme mostrado na Tabela 3.1.

A conversão de números da base 8 para a 2 é realizada de forma semelhante, no sentido inverso; substitui-se cada algarismo octal pelos seus 3 bits correspondentes (ver Tabela 3.1).

Exemplo 3.4

1) $(111010111)_2 = (\ \ )_8$

$(111)\ (010)\ (111)_2 = (727)_8$

7 2 7

2) $(1010011111)_2 = (\ \ )_8$

$(001)\ (010)\ (011)\ (111)_2 = (1237)_8$

1 2 3 7

3) $(327)_8 = (\ \ )_2$

$(011)\ (010)\ (111)_2 = (011010111)_2$ ou $(11010111)_2$

3 2 7

Obs.: Naturalmente, despreza-se o(s) zero(s) à esquerda do número.

4) $(673)_8 = (\ \ )_2$

$(110)\ (111)\ (011)_2 = (110111011)_2$

6 7 3

3.3.1.2 Entre as Bases 2 e 16

O procedimento de conversão entre números binários e hexadecimais (base 16) é idêntico ao da conversão entre as bases 2 e 8, exceto que, neste caso, a relação é $16 = 2^4$.

Desse modo, um algarismo hexadecimal é representado por 4 bits (ver Tabela 3.1). Converte-se um número binário em hexadecimal, dividindo-se este número em grupos de 4 bits da direita para a esquerda.

A conversão de hexadecimal para binário é obtida substituindo-se o algarismo hexadecimal pelos 4 bits correspondentes, de acordo com os valores indicados na Tabela 3.1.

Exemplo 3.5

1) $(1011011011)_2 = (\ \ )_{16}$

$(0010)\ (1101)\ (1011)_2 = (2DB)_{16}$

2 D B

2) $(10011100101101)_2 = (\ \ )_{16}$

$(0010)\ (0111)\ (0010)\ (1101)_2 = (272D)_{16}$

2 7 2 D

3) $(306)_{16} = (\ \ )_2$

$(0011)\ (0000)\ (0110)_2 = (1100000110)_2$

3 0 6

4) $(F50)_{16} = (\ \ )_2$

$(1111)\ (0101)\ (0000)_2 = (111101010000)_2$

F 5 0

3.3.1.3 Entre as Bases 8 e 16

O processo de conversão utiliza os mesmos princípios antes apresentados. No entanto, como a base de referência para as substituições de valores é a base 2, esta deve ser empregada como intermediária no processo. Ou seja, convertendo-se da base 8 para a base 16, deve-se primeiro efetuar a conversão para a base 2 (como mostrado nos subitens anteriores) e depois para a base 16. E o mesmo ocorre se a conversão for da base 16 para a base 8.

Exemplo 3.6

1) $(3174)_8 = (\ \ )_{16}$

Primeiro, converte-se o número da base 8 para a base 2:

$(011)\ (001)\ (111)\ (100)_2 = (011001111100)_2$

Em seguida, converte-se da base 2 para a base 16, separando-se os algarismos de 4 em 4, da direita para a esquerda:

$(0110)\ (0111)\ (1100) = (67C)_{16}$

6 7 C

2) $(254)_8 = (\ \ )_{16}$

$= (010)\ (101)\ (100)_2 = (010101100)_2$

$= (1010)\ (1100)_2 = (AC)_{16}$

3) $(2E7A)_{16} = (\ \ )_8$

$= (0010)\ (1110)\ (0111)\ (1010)_2 = (0010111001111010)_2 =$

$= (010)\ (111)\ (001)\ (111)\ (010)_2 = (27172)_8$

4) $(3C7)_{16} = (\ \ )_8$

$= (0011)\ (1100)\ (0111)_2 = (1111000111)_2 =$

$= (001)\ (111)\ (000)\ (111)_2 = (1707)_8$

3.3.2 Conversão de Números de uma Base B para a Base 10

A conversão de um número, representado em uma base B qualquer, para seu correspondente valor na base 10 é realizada empregando-se a Eq. 1.2. A melhor maneira de compreender o processo de conversão consiste na realização de alguns exemplos práticos, onde se indica, detalhadamente, a aplicação da referida equação.

Os exemplos apresentados referem-se apenas a números inteiros. No Apêndice A — Sistemas de Numeração, são detalhados os diversos processos de conversão de números inteiros e fracionários.

Exemplo 3.7

1) $(101101)_2 = (\ )_{10}$

Substituindo, na Eq. 1.2, as letras pelos valores do exemplo, teremos:

$b = 2$ (a base origem do número a ser convertido)

$n = 6$ (6 algarismos)

$n - 1 = 5$ (expoente do 1.° produto mais à esquerda)

$d_{n-1} = 1$ (algarismo mais à esquerda)

1.° produto: $d_{n-1} \times b^{n-1} = 1 \times 2^5$

Os demais produtos seguem a seqüência da Eq. 1.2, resultando em:

$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 =$

$= 32 + 0 + 8 + 4 + 0 + 1 = (45)_{10}$

2) $(27)_8 = (\ )_{10}$

Da mesma maneira, substitui-se na Eq. 1.2:

$b = 8$

$n = 2$

$n - 1 = 1$

$$\begin{array}{|c|c|c|c|c|c|c|} d_{n-1} & \times & B^{n-1} & + & d_0 & \times & B^0 \\ 2 & \times & 8^1 & + & 7 & \times & 8^0 \end{array}$$

$d_{n-1} = 2$

Valor total:

$2 \times 8^1 + 7 \times 8^0 = 16 + 7 = (23)_{10}$

3) $(2A5)_{16} = (\ )_{10}$

$2 \times 16^2 + 10 \times 16^1 + 5 \times 16^0 =$

$= 512 + 160 + 5 = (677)_{10}$

4) $(6734)_8 = (\ )_{10}$

$6 \times 8^3 + 7 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 =$

$= 3072 + 448 + 24 + 4 = (3548)_8$

5) $(27)_8 = (\ )_{10}$

$2 \times 8^1 + 7 \times 8^0 = 23_{10}$

Observação: No desenvolvimento foram suprimidos os produtos em que os algarismos eram 0, visto que o resultado seria também (0 3 2^8 etc.).

6) $(457)_9 = (\ )_{10}$

$4 \times 9^2 + 5 \times 9^1 + 7 \times 9^0 =$

$= 324 + 45 + 7 = (376)_{10}$

7) $(243)_5 = (\ )_{10}$

$2 \times 5^2 + 4 \times 5^1 + 3 \times 5^0 =$

$= 50 + 20 + 3 = (73)_{10}$

3.3.3 Conversão de Números Decimais para uma Base B

A conversão de números, representados na base 10, para seus valores equivalentes em uma base B qualquer é efetuada através de um processo inverso ao do subitem anterior (base B para base 10).

A conversão é obtida dividindo-se o número decimal pelo valor da base desejada; o resto encontrado é o algarismo menos significativo do valor na base B (mais à direita). Em seguida, divide-se o quociente encontrado pela base B; o resto é o algarismo seguinte (à esquerda); e assim, sucessivamente, vão-se dividindo os quocientes pelo valor da base até se obter quociente de valor zero. Em cada divisão, o resto encontrado é um algarismo significativo do número na nova base; o primeiro resto encontrado é o valor do algarismo menos significativo (mais à direita), e o último resto será o algarismo mais significativo (mais à esquerda).

Na realidade, o algoritmo de conversão pode ser definido com vários critérios de parada, tais como:

a) Enquanto o quociente for diferente de zero:
   - dividir dividendo por divisor;
   - extrair resto como algarismo e colocá-lo à esquerda do anterior;
   - repetir.

   Quando o quociente for igual a zero, parar.

b) Enquanto o dividendo for maior que o divisor:
   - dividir dividendo por divisor;
   - extrair resto como algarismo e colocá-lo à esquerda do anterior;
   - repetir.

Usar o dividendo (que agora é menor que o divisor) como último algarismo à esquerda (algarismo mais significativo).

No Apêndice A — Sistemas de Numeração, são detalhados os procedimentos de conversão de números inteiros e fracionários.

Exemplo 3.8

1) $(3964)_{10} = (\ \ )_8$

$3964/8 = 495$ $\quad$ $\text{resto}_0 = 4$ (algarismo menos significativo)

$495/8 = 61$ $\quad$ $\text{resto}_1 = 7$

$61/8 = 7$ $\quad$ $\text{resto}_2 = 5$

$7/8 = 0$ $\quad$ $\text{resto}_3 = 7$ (algarismo mais significativo)

O número é, então, $(7574)_8$.

2) $(483)_{10} = (\ \ )_8$

$483/8 = 60$ $\quad$ $\text{resto}_0 = 3$ (algarismo menos significativo, mais à direita)

$60/8 = 7$ $\quad$ $\text{resto}_1 = 4$

$7/8 = 0$ $\quad$ $\text{resto}_3 = 7$ (algarismo mais significativo, mais à esquerda)

O número é $(743)_8$.

Para verificar, façamos o processo inverso, isto é: converter $(743)_8$ para a base 10.

$7 \times 8^2 + 4 \times 8^1 + 3 \times 8^0 =$

$= 448 + 32 + 3 = (483)_{10}$

3) $(45)_{10} = (\ \ )_2$

$45/2 = 22$ $\quad$ $\text{resto}_0 = 1$ (algarismo menos significativo, mais à direita)
$22/2 = 11$ $\quad$ $\text{resto}_1 = 0$
$11/2 = 5$ $\quad$ $\text{resto}_2 = 1$
$5/2 = 2$ $\quad$ $\text{resto}_3 = 1$
$2/2 = 1$ $\quad$ $\text{resto}_4 = 0$
$1/2 = 0$ $\quad$ $\text{resto}_5 = 1$ (algarismo mais significativo, mais à esquerda)
O número é 101101_2.

4) $(97)_{10} = (\ \ )_2$

$97/2 = 48$ $\quad$ $\text{resto}_0 = 1$ (algarismo menos significativo)
$48/2 = 24$ $\quad$ $\text{resto}_1 = 0$
$24/2 = 12$ $\quad$ $\text{resto}_2 = 0$
$12/2 = 6$ $\quad$ $\text{resto}_3 = 0$
$6/2 = 3$ $\quad$ $\text{resto}_4 = 0$
$3/2 = 1$ $\quad$ $\text{resto}_5 = 1$
$1/2 = 0$ $\quad$ $\text{resto}_6 = 1$ (algarismo mais significativo)
O número é $(1100001)_2$.

5) $(2754)_{10} = (\ \ )_{16}$

$2754/16 = 172$ $\quad$ $\text{resto}_0 = 2$ $\quad$ algarismo 2_{16} (algarismo menos significativo)
$172/16 = 10$ $\quad$ $\text{resto}_1 = 12$ $\quad$ algarismo C_{16}
$10/16 = 0$ $\quad$ $\text{resto}_2 = 10$ $\quad$ algarismo A_{16} (algarismo mais significativo)
O número é $(AC2)_{16}$.

6) $(490)_{10} = (\ \ )_{16}$

$490/16 = 30$ $\quad$ $\text{resto}_0 = 10_{10}$ $\quad$ algarismo A_{16} (algarismo menos significativo)
$30/16 = 1$ $\quad$ $\text{resto}_1 = 14_{10}$ $\quad$ algarismo E_{16}
$1/16 = 0$ $\quad$ $\text{resto}_2 = 1_{10}$ $\quad$ algarismo 1_{16} (algarismo mais significativo)
O número é $(1EA)_{16}$.

É possível simplificar o processo de conversão de valores da base 2 para a base 10 e vice-versa. Para tanto, basta considerar o seguinte:

a) A Eq. 1.2 estabelece o valor de um número pela soma de produtos:

$d_{n-1} \times b^{n-1} + \ldots$

b) Cada produto é constituído de duas parcelas: a primeira é o algarismo correspondente à posição em que se encontra e a segunda é a potência da base, cujo índice indica a posição.

c) No caso de a base ser 2, os algarismos só podem assumir o valor 0 ou 1. Dessa forma, o resultado do produto somente pode ser 0 ou o próprio valor da potência de 2.

Exemplos:

$101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

O primeiro produto, 1×2^2, tem valor igual a $2^2 = 4$. Isto é, como o algarismo é 1, então 1×2^2 ou apenas 2^2 tem mesmo valor. No caso do segundo produto, 0×2^1 é igual a zero. O terceiro produto, igual ao primeiro, é 1×2^0 ou $2^0 = 1$.

d) As potências de 2, da direita para a esquerda, crescem da seguinte forma:

$2^0 = 1$ (potência zero, correspondente à posição mais à direita)

$2^1 = 2$; $2^2 = 4$; $2^3 = 8$; $2^4 = 16$ etc.

Ou seja:

... 6	5	4	3	2	1	0	←	posição
... 2^6	2^5	2^4	2^3	2^2	2^1	2^0	←	potência
... 64	32	16	8	4	2	1	←	valor

Em conseqüência, converter um número da base 2 para a base 10 consiste essencialmente em somar as potências de 2 correspondentes às posições onde o algarismo é igual a 1, desprezando as potências onde o algarismo é zero.

Exemplo 3.9

Efetuar as seguintes conversões:

1) $(110011)_2 = (\quad)_{10}$

5	4	3	2	1	0	←	posição
1	1	0	0	1	1	←	algarismo
2^5	2^4	—	—	2^1	2^0	←	potências válidas para somar
32	16	—	—	2	1	←	valores

Valor em base 10: $32 + 16 + 2 + 1 = (51)_{10}$

2) $(100111)_2 = (\quad)_{10}$

32	16	8	4	2	1	←	potências
1	0	0	1	1	1	←	algarismos

Valor em base 10: somam-se as potências válidas, correspondentes à posição onde o algarismo é 1.

$32 + 4 + 2 + 1 = (39)_{10}$

3.4 ARITMÉTICA NÃO-DECIMAL

Neste item serão apresentados procedimentos para realização das quatro operações aritméticas (adição, subtração, multiplicação e divisão) de números não-decimais (qualquer outro sistema de base diferente de 10), essencialmente os de base 2 e potência de 2, que interessam aos sistemas de computação.

Os números serão inteiros, sem limite de tamanho e positivos (sem sinal).

No Apêndice A — Sistemas de Numeração, são detalhados procedimentos para execução de operações aritméticas, com números binários, octais e hexadecimais, incluindo valores inteiros e fracionários, porém ainda sem sinal.

No Cap. 7, são detalhados procedimentos para execução de operações aritméticas com números positivos e negativos (inclusão do sinal nos números), inteiros e fracionários, bem como aqueles expressos na forma BCD (Binary Coded Decime). Os procedimentos estão relacionados ao processo efetivamente realizado no interior da unidade de processamento dos computadores. Já, neste capítulo, procura-se descrever procedimentos apenas matemáticos, para familiarizar o leitor com operações matemáticas não-decimais.

Finalmente, não se está levando em conta qualquer limite dos números, ou seja, a quantidade máxima de algarismos permitida para um dado número, o que é uma efetiva preocupação no caso dos computadores. Trata-se do problema de *overflow* ou estouro do limite, quando uma operação aritmética resulta em um valor acima do limite máximo possível (ver Cap. 7).

3.4.1 Aritmética Binária

3.4.1.1 Soma Binária

A operação de soma de dois números em base 2 é efetuada de modo semelhante à soma decimal, levando-se em conta, apenas, que só há dois algarismos disponíveis (0 e 1). Assim, podemos criar uma tabela com todas as possibilidades:

0 + 0 = 0	0 + 1 = 1
1 + 0 = 1	1 + 1 = 0, com "vai 1" ou 10_2

Exemplo 3.10 (adição)

a) Efetuar a soma 45_{10} e 47_{10}:		**b) Efetuar a soma 37_{10} e 87_{10}:**	
Decimal	**Binário**	**Decimal**	**Binário**
1	1 1111	11	111
45	101101	37	0100101
+ 47	+ 101111	+ 87	+ 1010111
92	1011100	124	1111100

Exemplo 3.11 (adição)

a) Efetuar a soma 27_{10} e 25_{10}:		**b) Efetuar a soma 11_{10} e 14_{10}:**	
Decimal	**Binário**	**Decimal**	**Binário**
1	11 11		111
27	11011	11	1011
+ 25	+ 11001	+ 14	+ 1110
52	110100	25	11001

Exemplo 3.12 (adição)

a) Efetuar a soma 357_{10} e 315_{10}:		**b) Efetuar a soma 99_{10} e 91_{10}:**	
Decimal	**Binário**	**Decimal**	**Binário**
1	1 1111111	11	1 11
357	101100101	99	1100011
+ 315	100111011	+ 91	1011011
672	1010100000	190	10111110

3.4.1.2 Subtração Binária

A subtração em base 2, na forma convencional, usada também no sistema decimal (minuendo − subtraendo = diferença), é relativamente mais complicada por dispormos apenas dos algarismos 0 e 1 e, dessa forma, 0 menos 1 necessita de "empréstimo" de um valor igual à base (no caso é 2), obtido do primeiro algarismo diferente de zero, existente à esquerda. Se estivéssemos operando na base decimal, o "empréstimo" seria de valor igual a 10.

Exemplo 3.13 (subtração)

a) Efetuar a subtração 101101 – 100111:

$$\begin{array}{r} 2\\ 002\\ 10\mathit{1}\mathit{1}01\\ -\ 100111\\ \hline 000110 \end{array}$$

A partir da direita para a esquerda, vamos executar a operação algarismo por algarismo (6 algarismos).

1) 1 − 1 = 0 (primeiro algarismo do resultado — mais à direita).
2) 0 − 1 não é possível. Então, retira-se 1 da ordem à esquerda (3.ª ordem a partir da direita), que fica com 1 − 1 = 0, e passa-se para a ordem à direita, o valor equivalente, que é 2, visto que 1 unidade de ordem à esquerda vale uma base de unidades (no caso: Base = 2) da ordem à direita.

   2 − 1 = 1 (segundo algarismo do resultado)
3) Agora tem-se 0 − 1 e, portanto, repete-se o procedimento do item anterior.

   2 − 1 = 1
4) 0 − 0 = 0
5) 0 − 0 = 0
6) 1 − 1 = 0

Resultado: 000110_2 ou simplesmente 110_2.

Exemplo 3.14 (subtração)

a) Efetuar a subtração 100110001 – 10101101:

$$\begin{array}{r} 1\\ 02\ \ 022\\ \mathit{1}001\mathit{1}0001\\ -\ 010101101\\ \hline 010000100 \end{array}$$

A partir da direita para a esquerda:

1) 1 − 1 = 0
2) 0 − 0 = 0
3) 0 − 1 não é possível. Retira-se 1 da 5.ª ordem, a partir da direita, ficando 2 unidades na 4.ª ordem. Dessas 2 unidades, retira-se 1 unidade para a 3.ª ordem (nesta 3.ª ordem ficam, então, 2), restando 1 unidade nesta 4.ª ordem.

   2 − 1 = 1
4) 1 − 1 = 0
5) 0 − 0 = 0
6) 1 − 1 = 0
7) 0 − 0 = 0
8) 0 − 1 não é possível. Retira-se 1 da ordem à esquerda, que fica com zero e passam-se 2 unidades para a direita.

   2 − 1 = 1
9) 0 − 0 = 0

Resultado: $(010000100)_2$

Exemplo 3.15 (subtração)

a) Efetuar a subtração 37 – 26:

Decimal	Binário
	1
	02202
37	100101
– 26	– 011010
11	001011

b) Efetuar a subtração 201 – 187:

Decimal	Binário
9	1121
1 $\cancel{1}$0	022022
2 0 1 +10	11001001
1 8 7	10111011
0 1 4	00001110

3.4.1.3 Multiplicação Binária

As regras para realização de multiplicação com números binários são exatamente iguais às das multiplicações decimais, com uma enorme vantagem sobre estas pelo fato de que só temos 2 algarismos em vez de 10. Desse modo, temos apenas:

0 × 0 = 0

0 × 1 = 0

1 × 0 = 0

1 × 1 = 1

Enquanto na multiplicação decimal temos uma tabela com 100 operações, do tipo:

1 × 2 = 2; 2 × 7 = 14; 5 × 6 = 30 etc.

Para melhor entendimento sobre o assunto, basta observar alguns exemplos, com a descrição detalhada de cada passo e incluindo em todos os exemplos a operação em decimal e em binário.

Exemplo 3.16 (multiplicação)

Efetuar a multiplicação 6 × 5:

Decimal	Binário	
6	110	← multiplicando
× 5	× 101	← multiplicador
30	110	← produtos parciais
	000	
	110	
	11110	← resultado

O procedimento consiste em multiplicar cada algarismo do multiplicador pelos algarismos do multiplicando, resultando em sucessivos produtos parciais, tantos quantos forem os algarismos do multiplicador. No Exemplo 3.16 são três algarismos e, portanto, temos três produtos parciais.

Cada produto parcial é colocado de modo a se posicionar uma casa para a esquerda do produto anterior, isto é, há um deslocamento do 2.° produto para a esquerda em relação ao 1.° produto e há um deslocamento à esquerda do 3.° produto em relação ao 2.° produto.

Em seguida, os três produtos são somados produzindo o resultado desejado.

No caso de sistemas binários, o procedimento é ainda mais simples porque os produtos parciais só podem ser zero (se o algarismo do multiplicando for zero) ou o próprio valor do multiplicador (se o algarismo do multiplicando for um).

Exemplo 3.17 (multiplicação)

Efetuar a multiplicação 21 × 13:

Decimal **Binário**

$$\begin{array}{r}
21 \\
\times\ 13 \\
\hline
63 \\
+21 \\
\hline
273
\end{array}
\qquad\qquad
\begin{array}{rl}
10101 & \leftarrow \text{multiplicando} \\
\times\ 1101 & \leftarrow \text{multiplicador} \\
\cline{1-1}
10101 & \leftarrow \text{4 produtos parciais,} \\
+00000 & \ \text{cada um deslocado} \\
10101 & \ \text{1 casa para a} \\
10101 & \ \text{esquerda} \\
\cline{1-1}
100010001 & \leftarrow \text{resultado}
\end{array}$$

Exemplo 3.18 (multiplicação)

Efetuar a multiplicação 18 × 4:

Decimal **Binário**

$$\begin{array}{r}
18 \\
\times\ 4 \\
\hline
72
\end{array}
\qquad\qquad
\begin{array}{r}
10010 \\
\times\ 100 \\
\hline
00000 \\
00000 \\
10010 \\
\hline
1001000
\end{array}$$

Neste exemplo, bastaria acrescentarmos dois zeros à direita do multiplicando e teríamos o mesmo resultado da operação completa.

multiplicando: 10010

mais dois zeros: 1001000 ← resultado

Isso acontece porque o multiplicador é constituído do algarismo 1 (repetição do multiplicando) seguido de dois zeros. O produto parcial de cada multiplicador por zero é igual a zero e, portanto, a soma com o multiplicando resulta no próprio valor do multiplicador, porém deslocada uma ordem para a esquerda, o que significa acréscimo de um zero à direita.

3.4.1.4 Divisão Binária

O procedimento matemático para realização de uma operação de divisão com números binários é semelhante ao procedimento para a mesma operação com valores decimais.

O procedimento compreende a manipulação de quatro elementos:

dividendo — valor a ser dividido

divisor — valor que deve estar contido *n* vezes no dividendo e que, então, se deseja saber qual o valor de *n*

quociente — quantidade de vezes que o divisor se repete no dividendo (valor de *n*)

resto — caso a divisão não seja exata, isto é, o divisor vezes *n* não seja igual ao dividendo, a diferença é chamada de resto.

Vamos descrever o processo na base 10 para entendermos bem cada passo e, em seguida, exemplificar na base 2, seguindo os mesmos procedimentos.

Exemplo 3.19

1) 35/5 = 7, com resto = 0 e

2) 37/5 = 7, com resto = 2.

Nestes exemplos, o dividendo é 35 e 37, os divisores são, em ambos os casos, 5, o quociente é igual a 7 em ambos os casos e o resto é, respectivamente, 0 e 2.

$$\begin{array}{r|l} 35 & 5 \\ \cline{2-2} -35 & 7 \\ \hline 0 & \end{array} \qquad \begin{array}{r|l} 37 & 5 \\ \cline{2-2} -35 & 7 \\ \hline 2 & \end{array}$$

Procedimento:

a) Verifica-se quantas vezes o divisor cabe no dividendo por tentativa.

b) Iniciam-se, mentalmente ou por qualquer outro método que o leitor encontre confortável, as tentativas tais como:

   2 × 5 = 10, 3 × 5 = 15, 4 × 5 = 20 (todos menores que 35)

   E prossegue-se: 5 × 5 = 25, 6 × 5 = 30, 7 × 5 = 35 e 8 × 5 = 40. Como 40 é maior que 35 (ou 37, no caso do segundo exemplo) o valor escolhido para quociente é igual a 7.

c) Subtrai-se de 35 (dividendo) o valor resultante da multiplicação do quociente pelo divisor (7 × 5), encontrando-se um valor que é o resto da divisão. No primeiro exemplo, o valor é zero, 35 − 35 = 0, e no segundo exemplo é 2, 37 − 35 = 2.

d) O resto da divisão deve sempre ser um valor igual, no máximo, ao divisor menos 1. No exemplo, ele deverá ser, no máximo, igual a 4, pois se ele fosse 5, isso significaria que o quociente poderia ser maior, já que o divisor (valor 5) ainda cabe no dividendo.

Vejamos um exemplo de divisão binária.

Exemplo 3.20 (divisão)

Efetuar a divisão $(1001)_2$ por $(101)_2$:

$(1001)_2/(101)_2$

No caso da divisão binária o procedimento se torna mais simples, visto que cada algarismo do quociente só pode ser 1 (quando o divisor é menor — apenas 1 vez — que o dividendo ou parte dele) ou zero (caso contrário).

No exemplo acima, 101 é menor e cabe apenas 1 vez em 1001. O quociente é, então, 1 e

$$\begin{array}{r} 1001 \\ -\ 101 \\ \hline 0100 \end{array}$$

o resto é $(100)_2$.

Em decimal, $(1001)_2 = 9_{10}$, $(101)_2 = 5_{10}$ e $(100)_2 = 4_{10}$. Ou seja, 9/5 = 1, resto 4.

Vejamos em seguida um exemplo de operação de divisão binária com dividendo de valor bem maior que o divisor de modo que ocorram divisões parciais.

Exemplo 3.21 (divisão)

Efetuar a divisão 101010_2 por 101_2:

$(101010)_2/(110)_2$

a) Em primeiro lugar, verifica-se que valor (que quantidade de algarismos) é suficientemente maior que o divisor, de modo que o primeiro algarismo do quociente seja 1.

   No exemplo utilizado, o valor 1010 (quatro primeiros algarismos da esquerda para a direita) é maior uma vez que o divisor. Assim, temos inicialmente

```
 101010|_110
 - 110    1
 ------
   100
```

b) Em seguida, subtrai-se de 1010 (parte utilizada do dividendo) o valor 110 (que é 1 × 110), ou seja, quociente, 1, vezes divisor, 110, encontrando-se como resto parcial 100.

c) Efetua-se nova divisão, utilizando-se como novo dividendo o valor do resto parcial 100 acrescido de um algarismo do dividendo completo, sendo, no caso, o algarismo 1.

O novo dividendo será 1001, que contém 1 vez o divisor, 110. E assim teremos nova divisão parcial

```
  1001|_110
 - 110  1
 -----
   011
```

d) Repete-se pela 3.ª vez o processo, dividindo-se 110 (novo dividendo, formado pelo resto parcial 11 acrescido do último algarismo do dividendo completo, 0, por 110. Encontra-se quociente 1 e resto parcial 000. A divisão está completada.

```
  110|_110
 -110  1
 ----
  000
```

A operação completa fica assim:

```
 101010|_110
 - 110    111
 ------
   1001
 -  110
   ----
   0110
 -  110
   ----
    000
```

Exemplo 3.22 (divisão)

a) Efetuar a divisão 37_{10} por 4,0:

```
        Decimal                 Binário

         37/4                 100101/100

                              100101|_100
         37|_4                 100    1001
         36  9               ------
         --                    0101
          1                     100
                               ----
                                001
```

a) Divide-se 100 (menor valor do dividendo que é ainda igual ou maior que o divisor) por 100 (divisor), encontrando-se quociente 1, com resto parcial 0 (100 − 100).

b) Acrescenta-se ao resto 0 tantos algarismos do dividendo (um a um da esquerda para a direita) quantos necessários para que o valor obtido seja igual ou maior que o divisor. A cada algarismo selecionado e não suficiente acrescenta-se um zero ao quociente.

c) No exemplo, foram selecionados os algarismos 101 (acrescentou-se 00 ao quociente, para os algarismos 10 que formaram o valor 010, ainda menor que o divisor 100. Finalmente acrescentou-se 1 (último algarismo disponível do dividendo), resultando 101, neste caso superior a 100. Então, o quociente foi acrescido de 1.

3.4.2 Aritmética Octal (Em Base 8)

O sistema binário, por ser constituído de tão poucos algarismos diferentes em sua base (0 e 1), causa o fato de os números serem na maioria das vezes constituídos de uma enorme quantidade de algarismos. Na realidade, podemos generalizar a regra de que "quanto menor o valor da base, maior é a quantidade de algarismos de um número naquela base".

Assim, por exemplo, o número 9 na base 10 só possui um algarismo, porém na base 6 é constituído de dois algarismos, 13, e na base 2 necessita de quatro algarismos para sua representação:

$9_{10} = 13_6 = 1001_2$

Quanto maior o número, mais rapidamente cresce a quantidade de dígitos binários (bits) necessários para essa representação:

$(1011001111011101)_2 = (46045)_{10} = (B3DD)_{16}$

Além disso, os sistemas de base menor (e o sistema binário é o menor de todos) ainda possuem um outro inconveniente. A quantidade de algarismos disponíveis na base sendo pequena, há pouca diferença entre os algarismos e, sendo muitos, acarreta dificuldade de percepção do usuário. Basta ver o exemplo anterior, onde o valor binário possui 16 algarismos com variação apenas entre 0s e 1s, muito mais complicado de compreensão por parte do leitor do que o valor 46045, com apenas cinco algarismos e vários símbolos diferentes (o 4, o 6, o 5 e o 0) ou mais ainda, na base 16, com o valor B3DD, com apenas 4 algarismos.

O nosso sistema visual distingue melhor variações acentuadas entre elementos (por exemplo, entre B, 3, D, 4, 5, 6 etc.) do que diferenças mínimas, tais como apenas 0s e 1s. Por isso, distinguimos melhor as diferenças entre objetos coloridos do que estes mesmos objetos em preto e branco, apenas distinguidos por tons diferentes de preto.

Por esta razão, apesar de internamente nos computadores o sistema ser essencialmente binário, costuma-se empregar bases mais elevadas para representar externamente os valores armazenados ou manipulados pelos computadores.

Utiliza-se com freqüência as bases 8 e 16 por serem bases maiores, e a conversão da base 2 para elas, e vice-versa, é mais rápida que para a base 10. Atualmente, a base 16 é a base mais usada para representar, em manuais, vídeos etc., estes valores que estão internamente em binário.

Com este propósito, vamos apresentar alguns aspectos da aritmética octal e hexadecimal, apenas as operações de adição e subtração, visto que as outras não se aplicam para o fim a que se destinam.

Para finalizar e consolidar o assunto, apresentamos alguns exemplos de aritmética em qualquer outra base não-decimal nem potência de 2.

Exemplo 3.23 (adição)

Efetuar a soma $(3657)_8 + (1741)_8$:

$$\begin{array}{rl} 111 & \leftarrow \text{“vai 1} \\ 3657 & \leftarrow 1.^{\text{a}}\ \text{parcela} \\ +1741 & \leftarrow 2.^{\text{a}}\ \text{parcela} \\ \hline 5620 & \end{array}$$

Da direita para a esquerda, temos para cada um dos 4 algarismos:

1) $7 + 1 = 8$

Como não há algarismo 8 na base 8, emprega-se o conceito posicional, isto é, 8 unidades de uma ordem valem 1 unidade da ordem imediatamente à esquerda. Então: fica $0 = 8 - 8$ e "vai 1" para a esquerda.

2) 1 (vai 1 vindo da ordem à direita) + 5 + 4 = 10

   Utilizando o mesmo conceito anterior, temos:

   10 − 8 = 2 e "vai 1" (que é igual a 8).

3) 1 (vai 1) + 6 + 7 = 14

   14 − 8 = 6 e "vai 1"

4) 1 + 3 + 1 = 5 Não há "vai 1" porque não se excede 7.

   Resultado: 5620_8

Exemplo 3.24 (adição)

Efetuar a soma $(443)_8 + (653)_8$:

$$\begin{array}{rl} 11 & \leftarrow \text{“vai1} \\ 443 & \leftarrow 1.^{\text{a}}\ \text{parcela} \\ +653 & \leftarrow 2.^{\text{a}}\ \text{parcela} \\ \hline 1316 & \end{array}$$

Da direita para a esquerda, para cada um dos 3 algarismos:

1) 3 + 3 = 6

   Como 6 é um algarismo válido da base 8, não há "vai 1".

2) 4 + 5 = 9

   Então: 9 − 8 = 1 e "vai 1" (que correspondem às 8 unidades em excesso).

3) 1 + 4 + 6 = 11

   Então: 11 − 8 = 3 e "vai 1".

4) 1 + 0 = 1

   Resultado: 1316_8

Exemplo 3.25 (subtração)

Efetuar a subtração $(7312)_8 - (3465)_8$:

$$\begin{array}{rl} 88 & \\ 6208 & \leftarrow \text{empréstimos} \\ \cancel{7}\cancel{3}\cancel{1}2 & \leftarrow 1.^{\text{a}}\ \text{parcela} \\ -3465 & \leftarrow 2.^{\text{a}}\ \text{parcela} \\ \hline 3625 & \end{array}$$

Da direita para a esquerda, temos para cada um dos 4 algarismos:

1) 2 − 5 não é possível. Então, retira-se 1 unidade da ordem à esquerda, a qual vale uma base de unidades (no caso base = 8) da direita, somando-se ao valor 2.

   8 + 2 = 10 − 5 = 5

2) 1 − 1 = 0 − 6 não é possível. Então, retira-se 1 unidade da esquerda (que fica com 3 − 1 = 2 unidades), passando 8 para a direita, o que fica 8 + 0 = 8.

   8 − 6 = 2

3) 3 − 1 = 2 − 4 não é possível. Então, retira-se 1 da esquerda (7 − 1 = 6), passando 8 unidades para a direita.

$8 + 2 = 10 - 4 = 6$

4) $7 - 1 = 6 - 3 = 3$

Resultado: 3625_8

3.4.3 Aritmética Hexadecimal (Em Base 16)

Já mencionamos anteriormente que a aritmética com valores expressos em algarismos hexadecimais segue as mesmas regras para qualquer base: somar ou subtrair algarismo por algarismo, utilizando-se de "vai x" na casa à esquerda (e somando-o com as parcelas seguintes à esquerda) ou de "empréstimo" (como nas subtrações em qualquer outra base), e assim por diante.

Exemplo 3.26 (adição)

Efetuar a soma $(3A943B)_{16} + (23B7D5)_{16}$:

$$\begin{array}{rl}
1\ \ 11 & \leftarrow \text{“vai1”} \\
3A943B & \leftarrow 1.^{\text{a}}\ \text{parcela} \\
+23B7D5 & \leftarrow 2.^{\text{a}}\ \text{parcela} \\ \hline
5E4C10 &
\end{array}$$

Da direita para a esquerda, temos para cada um dos 6 algarismos:

1) $B = 11_{10} + 5_{16} = 16_{10}$

   Como 16_{10} não é um algarismo válido da base 16 (o maior algarismo, F, tem valor $= 15_{10}$), então usa-se o princípio posicional, substituindo 16 unidades da ordem da direita por 1 unidade da ordem à esquerda (vai 1).

   $B + 5 = 0$ e vai 1

2) $1 + 3 + D = 1 + 3 + 13 = 17_{10}$

   $17_{10} = 16$ (vai 1 para a esquerda) $+ 1$

3) $1 + 4 + 7 = 12_{10}$

   12_{10} equivale ao algarismo C_{16}. Coloca-se C como resultado e não há "vai 1".

4) $9 + B = 9 + 11 = 20_{10}$

   $20 = 16$ (vai 1 para a esquerda) $+ 4$. Coloca-se 4 como resultado e "vai 1" para a esquerda.

5) $1 + A + 3 = 1 + 10 + 3 = 14_{10}$

   14_{10} equivale ao algarismo E_{16}. Coloca-se E como resultado e não há "vai 1".

6) $3 + 2 = 5$. Coloca-se 5 como resultado e não há "vai 1".

   Resultado: $5E4C10_{16}$

Exemplo 3.27 (subtração)

Efetuar a subtração $(4C7BE8)_{16} - (1E927A)_{16}$:

$$\begin{array}{cccccc}
4 - 1 = 3 & C - 1 = B + 16 = 27 & & & E - 1 = D & 8 + 16 = 24 \\
\cancel{4} & \cancel{C} & 7 + 16 = 23 & B & \cancel{E} & 8 \\
-1 & E & 9 & 2 & 7 & A \\ \hline
2 & D & E & 9 & 6 & E
\end{array}$$

Da direita para a esquerda, para cada um dos 6 algarismos:

1) 8 − A não é possível. Retira-se, então, 1 unidade da ordem à esquerda (E − 1 = D), passando 16 unidades (valor igual ao da base) para a direita, as quais são somadas ao valor existente, 8.

   $16 + 8 = 24 - A = 24 - 10 = 14_{10}$, equivalente ao algarismo E_{16}

2) D − 7 = 13 − 7 = 6

3) B − 2 = 11 − 2 = 9

4) 7 2 9 não é possível. Retira-se 1 unidade da ordem à esquerda (C − 1 = B), passando 16 unidades para a direita, as quais são somadas ao valor existente, 7.

   $16 + 7 = 23 - 9 = 14_{10}$, equivalente ao algarismo E_{16}

5) C − E não é possível. Retira-se 1 unidade da ordem à esquerda (4 − 1 = 3), passando 16 unidades para a direita, as quais são somadas ao valor existente, $B_{16} = 11_{10}$.

   $16_{10} + B_{16} = 16_{10} + 11_{10} = 27 - 14 = 13_{10}$, equivalente ao algarismo D_{16}

6) 3 − 1 = 2

   Resultado: $2DE96E_{16}$

Observação: No Apêndice A é mostrado com mais detalhe como se efetuam operações de subtração e soma com algarismos na base 16, como, por exemplo, a soma do algarismo A_{16} com o algarismo C_{16}. Além disso, são descritas também as operações aritméticas de multiplicação e divisão.

EXERCÍCIOS

1) Converter os seguintes valores decimais em valores binários equivalentes (conversão de base 10 para base 2):

   a) 329
   b) 284
   c) 473
   d) 69
   e) 135
   f) 215
   g) 581
   h) 197

2) Converter os seguintes valores binários em valores decimais equivalentes (conversão de base 2 para base 10):

   a) 11011101010
   b) 11001101101
   c) 10000001111
   d) 11101100010
   e) 111001101001
   f) 111111000011
   g) 101100011000
   h) 100000000110

3) Converter os seguintes valores decimais em valores octais equivalentes (conversão de base 10 para base 8):

   a) 177
   b) 254
   c) 112
   d) 719
   e) 343
   f) 27
   g) 821
   h) 197

4) Converter os seguintes valores decimais em valores binários equivalentes (conversão de base 10 para base 2):

   a) 417
   b) 113
   c) 819
   d) 77
   e) 251
   f) 769
   g) 180
   h) 27

5) Converter os seguintes valores binários em valores decimais equivalentes (conversão de base 2 para base 10):

a) 1100011
b) 10101111101
c) 11000011001
d) 101101
e) 1000000011
f) 111100011110110
g) 1100100001
h) 1101110

6) Converter os seguintes valores decimais em valores octais equivalentes (conversão de base 10 para base 8):

a) 917
b) 779
c) 610
d) 593
e) 325
f) 216
g) 413
h) 521

7) Converter os seguintes valores octais em valores decimais equivalentes (conversão de base 8 para base 10):

a) 405
b) 477
c) 237
d) 46
e) 705
f) 173
g) 201
h) 452

8) Converter os seguintes valores decimais em valores hexadecimais equivalentes (conversão de base 10 para base 16):

a) 447
b) 544
c) 223
d) 71
e) 622
f) 97
g) 121
h) 297

9) Converter os seguintes valores hexadecimais em valores decimais equivalentes (conversão de base 16 para base 10):

a) 3A2
b) 33B
c) 621
d) 99
e) 1ED4
f) 7EF
g) 22C
h) 11OA

10) Converter os seguintes valores octais em valores decimais equivalentes (conversão de base 8 para base 10):

a) 2136
b) 1741
c) 613
d) 546
e) 120
f) 317
g) 720
h) 665

11) Converter os seguintes valores decimais em valores hexadecimais equivalentes (conversão de base 10 para base 16):

a) 2173
b) 1325
c) 743
d) 212

e) 681

f) 937

g) 1480

h) 1671

12) Converter os seguintes valores hexadecimais em valores decimais equivalentes (conversão de base 16 para base 10):

a) 21A7

b) 1BC9

c) 27D

d) E5F

e) 2351

f) 19AE

g) ACEF

h) 214B

13) Efetuar as seguintes conversões de base:

a) $37421_8 = (\ \ )_{16}$

b) $14A3B_{16} = (\ \ )_{10}$

c) $11011100011_2 = (\ )_{16}$

d) $2BEF5_{16} = (\ \ )_8$

e) $5331_8 = (\ \ )_2$

f) $100011011_2 = (\ \ )_8$

g) $217_{10} = (\ \ )_7$

h) $413_8 = (\ \ )_2$

14) Efetuar as seguintes somas:

a) $31752_8 + 6735_8 =$

b) $37742_8 + 26573_8 =$

c) $2A5BEF_{16} + 9C829_{16} =$

d) $356_7 + 442_7 =$

e) $1100111101_2 + 101110110_2 =$

f) $211312_4 + 121313_4 =$

g) $3645_8 + 2764_8 =$

h) $110011110_2 + 11011111_2 =$

15) Efetuar as seguintes operações de subtração:

a) $64B2E_{16} - 27EBA_{16} =$

b) $2351_8 - 1763_8 =$

c) $543_6 - 455_6 =$

d) $43321_5 - 2344_5 =$

e) $11001000010_2 - 1111111111_2 =$

f) $10001101000_2 - 101101101_2 =$

g) $43DAB_{16} - 3EFFA_{16} =$

h) $100010_2 = 11101_2 =$

16) Efetuar as seguintes conversões de base:

a) $2317_8 = (\ \ )_2$

b) $1A45B_{16} = (\ \ )_8$

c) $3651_{16} = (\ \ )_2$

d) $11001011011011_2 = (\ \ )_8$

17) Efetuar as seguintes somas:

a) $3251_8 + 2167_8 =$

b) $2EC3BA_{16} + 7C35EA_{16} =$

c) $1011101_2 + 1111001_2 =$

d) $1110000101_2 + 1000011111_2 =$

e) $312321_4 + 112213_4 =$

f) $2AC79_{16} + B7EEC_{16} =$

g) $2748E_{16} + FA7B5_{16} =$

h) $217_8 + 173_8 =$

18) Efetuar as seguintes operações de subtração:

a) $110000001101_2 - 10110011101_2 =$

b) $35A3_{16} - 2FEC_{16} =$

c) $37425_8 - 14766_8 =$

d) $1001001_2 - 111100_2 =$

19) Quantos números inteiros positivos podem ser representados em uma base B, cada um com *n* algarismos significativos?

20) A partir do valor binário 110011, escreva os cinco números que se seguem em seqüência.

21) A partir do valor binário 101101, escreva seis números, saltando de 3 em 3 números, de forma crescente.

22) A partir do valor octal 1365, escreva os oito números que se seguem em seqüência.

23) A partir do valor octal 3745, escreva os oito números pares seguintes.

24) A partir do valor hexadecimal 2BEF9, escreva os 12 números que se seguem em seqüência.

25) A partir do valor hexadecimal 3A57, escreva os 10 números subseqüentes, saltando de quatro em quatro valores (por exemplo, o 1.° subseqüente é 3A5B).

26) A maioria das pessoas só pode contar até 10 utilizando seus dedos. Entretanto, quem trabalha com computador pode fazer melhor. Se você imaginar cada um dos seus dedos como um dígito binário, convencionando que o dedo estendido significa o algarismo 1 e o recolhido significa 0, até quanto você poderá contar usando as duas mãos?

27) Supondo um sistema posicional de numeração de base 4, determine, a partir da operação de adição a seguir, os valores de A, B, C e D:

$$\begin{array}{r} \text{BADB} \\ +\,\text{DDDC} \\ \hline \text{BCDCB} \end{array}$$

28) Suponha um sistema posicional de base 6. Determine os valores de A, B, C, D, E e F:

$$\begin{array}{r} \text{ADCFA} \\ \text{BABDF} \\ \hline \text{CFEDFB} \end{array}$$

29) Efetue as seguintes operações aritméticas na base indicada para o resultado:

a) $FEFE_{16} + 1110100100011110_2 = (\ \ )_8$

b) $7374_8 + 313202_4 = (\ \ )_{16}$

c) $384_{10} + 512_{16} = (\ \ )_{16}$

d) $532_6 + 101_8 = (\ \ )_{16}$

e) $10011101_2 + 376_8 = (\ \ )_{16}$

f) $3E54_{16} + 1257_8 = (\ \ )_8$

g) $10110110101_2 + 2FE_{16} = (\ \ )_8$

h) $1374_{10} + 11011011110111_2 = (\ \ )_8$

30) Expresse o valor decimal 100 em todas as bases entre 2 e 9 (inclusive).

31) Quantos números diferentes podem ser criados por um conjunto de quatro chaves, cada uma podendo gerar três diferentes algarismos?

32) Quantos números binários diferentes podem ser armazenados em memórias com espaço de armazenamento de seis dígitos cada uma?

33) Quantos números diferentes podem ser criados cada um possuindo três algarismos?

34) Quantos números binários diferentes podem ser criados cada um possuindo oito algarismos?

35) Qual é o valor decimal equivalente ao maior número de sete algarismos que pode existir na base 2?

36) Em cada uma das seguintes equivalências, ache o valor da base *b*, na qual o número à direita está expresso:

a) $496_{10} = 1306_b$

b) $249_{10} = 13B_b$

c) $1248_{16} = 11110_b$

d) $1248_{16} = 1021020_b$

37) Um hodômetro hexadecimal mostra o número 5ECFC. Quais são as seis próximas leituras?

38) Um hodômetro hexadecimal mostra o número A3FF. Qual é a leitura seguinte? Após rodar alguns quilômetros, o hodômetro apresenta a seguinte leitura: A83C. Quanto foi andado? (Dê a resposta em hexadecimal e em decimal.)

39) Converter os seguintes números, representados em base 6, para seus valores equivalentes em base 9.

a) 24_6

b) 144_6

c) 2354_6

d) 555_6

e) 3144_6

f) 211_6

40) Converter os seguintes números de uma base B para outra base B indicada:

a) $234_6 = (\quad)_8$

b) $1321_4 = (\quad)_7$

c) $431_5 = (\quad)_9$

d) $246458_9 = (\quad)_4$

e) $4452_6 = (\quad)_7$

f) $2112_3 = (\quad)_5$

41) Complete a tabela abaixo:

Decimal	**Binário**	**Octal**	**Hexadecimal**
37			
	11001101		
		356	
			1A4C
	10001101		
117			
			2A5B
		457	

42) Efetue as seguintes operações aritméticas:

a) $(101)_2 \times (111)_2 = (\quad)_2$

b) $(11101)_2 \times (1010)_2 = (\quad)_2$

c) $(11001110)_2 / (1101)_2 = (\quad)_2$

d) $(111110001)_2 \times (10011)_2 = (\quad)_2$

e) $(100100011)_2 / (11101)_2 = (\quad)_2$

f) $(1101101)_2 / (100)_2 = (\quad)_2$

g) $(111000001) \times (101001)_2 = (\quad)_2$

43) A soma dos dígitos mais e menos significativo (mais à esquerda e mais à direita, respectivamente) de um mínimo inteiro de três dígitos é igual a 10. A soma de todos os três algarismos é 18. O resultado do divisor do número pelo algarismo menos significativo (mais à direita) é igual a 171. Qual é o valor do número?

44) Se um número binário é deslocado uma ordem para a esquerda, isto é, cada um de seus bits move-se uma posição para a esquerda e um zero é inserido na posição mais à direita, obtém-se um novo número. Qual é a relação matemática existente entre os dois números?

45) A soma de dois números binários é 101000 e a diferença entre eles é igual a 1010. Quais são os números binários?

4

Conceitos da Lógica Digital

4.1 INTRODUÇÃO

Conforme já foi explicado anteriormente, um computador digital é uma máquina projetada para armazenar e manipular informações representadas apenas por algarismos ou dígitos e que só podem assumir dois valores distintos, 0 e 1, razão por que (ver Cap. 1) é chamado *computador digital, sistema digital* ou simplesmente *máquina digital binária.* Como na prática não há máquinas digitais não-binárias, como, por exemplo, máquinas digitais decimais, é mais usual simplificar-se o termo, usando apenas *computador digital* (a palavra binário fica implícita).

A informação binária (valores 0 ou 1) é representada em um sistema digital por quantidades físicas, sinais elétricos, os quais são gerados e mantidos internamente ou recebidos de elementos externos, em dois níveis de intensidade, cada um correspondente a um valor binário (há outras formas de armazenamento de bits internamente em um computador, como campo magnético e sinais óticos).

A Fig. 4.1 mostra um exemplo de valores elétricos de sinais binários, sendo escolhido um sinal de +3 V para representar o bit 1 e +0,5 V para representar o valor binário 0. Como se observa na figura, cada valor tem uma faixa de tolerância, tendo em vista que nenhum sinal é sempre absolutamente preciso em seu valor.

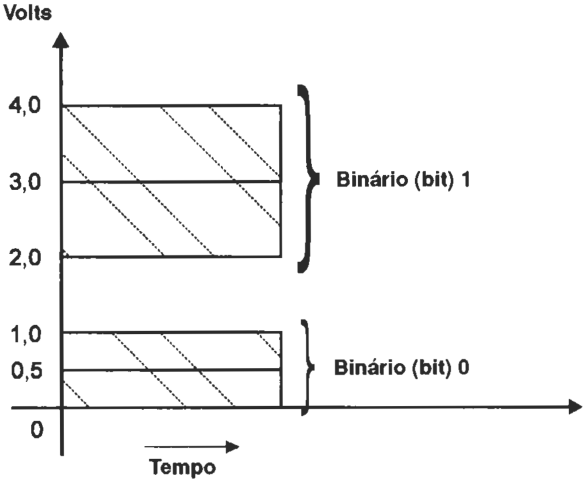


Figura 4.1 Exemplo de um sinal binário.

Internamente, um computador é constituído de elementos eletrônicos, como resistores, capacitores e principalmente transistores. Nesses computadores, os transistores são, em geral, componentes de determinados circuitos eletrônicos que precisam armazenar os sinais binários e realizar certos tipos de operações com eles. Esses circuitos, chamados circuitos digitais, são formados de pequenos elementos capazes de manipular grandezas apenas binárias. Os pequenos elementos referidos são conhecidos como *portas* (*gates*) lógicas, por permitirem ou não a passagem desses sinais, e os circuitos que contêm as portas lógicas são conhecidos como *circuitos lógicos*.

Uma *porta* (*gate*) é, então, um elemento de hardware (é um circuito eletrônico) que recebe um ou mais sinais de entrada e produz um sinal de saída, cujo valor é dependente do tipo de regra lógica estabelecida para a construção do referido circuito.

Em resumo, um computador digital é construído, então, contendo uma infinidade de circuitos lógicos ou portas, convenientemente distribuídos e organizados, de modo que alguns servirão para armazenamento de valores, outros permitirão e controlarão o fluxo de sinais entre os componentes e outros, ainda, serão utilizados para realizar operações matemáticas.

O projeto de circuitos digitais e a análise de seu comportamento em um computador podem ser realizados através do emprego de conceitos e regras estabelecidas por uma disciplina conhecida como *Álgebra de Chaveamentos* (*Switching Algebra*), que é um ramo da álgebra booleana ou álgebra moderna.

Neste capítulo vamos apresentar noções sobre portas lógicas e as operações que podem ser realizadas com estes elementos, bem como alguns aspectos de lógica digital que sejam considerados importantes para a compreensão do livro. Um pouco mais de profundidade sobre álgebra booleana e lógica digital pode ser encontrado especificamente no item 4.4.

4.2 PORTAS E OPERAÇÕES LÓGICAS

Uma *porta lógica* (*gate*) é um circuito eletrônico, portanto uma peça de hardware, que se constitui no elemento básico e mais elementar de um sistema de computação. Grande parte do hardware do sistema é fabricado através da adequada combinação de milhões desses elementos, como a UCP, memórias principal e cache, interfaces de E/S e outros.

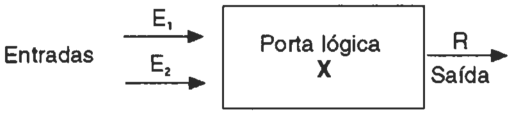
Há diversos tipos bem definidos de portas lógicas, cada uma delas capaz de implementar uma operação ou função lógica específica. Uma *operação lógica* (de modo semelhante a uma operação algébrica) realizada sobre um ou mais valores lógicos produz um certo resultado (também um valor lógico), conforme a regra definida para a específica operação lógica (ver Fig. 4.2).

Assim como na álgebra comum (que estudamos no 2.º grau), é necessário definir símbolos matemáticos e gráficos para representar as operações lógicas (e os operadores lógicos). A Fig. 4.3 mostra os símbolos matemáticos e gráficos referentes às operações lógicas (portas) que iremos analisar neste item. No item 4.4 mostraremos os símbolos das demais portas.

Como já citado anteriormente, uma operação lógica produz um resultado que pode assumir somente dois valores, 0 ou 1, os quais são relacionados na álgebra booleana às declarações FALSO (F = bit 0) ou VERDADEIRO (V = bit 1). Se as variáveis de entrada só podem assumir os valores F (falso) = 0 ou V (verdadeiro) = 1, e se o resultado também, então podemos definir previamente todos os possíveis valores de resultado de uma dada operação lógica conforme a combinação possível de valores de entrada. Essas possibilidades são representadas de forma tabular, e o conjunto se chama *Tabela Verdade*. Cada operação lógica possui sua própria *tabela verdade*, estabelecida de acordo com a regra que define a respectiva operação lógica.

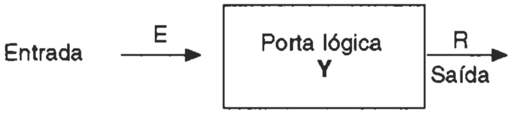
Uma *tabela verdade* tem, então, tantas linhas de informação quantas são as possíveis combinações de valores de entrada, o que pode variar conforme a quantidade de diferentes valores de entrada que se tenha.

Assim, se tivermos apenas um valor de entrada, então a saída só pode assumir dois valores (já que a variável de entrada só pode assumir dois valores distintos, F = 0 ou V = 1) e, nesse caso, a tabela verdade teria duas linhas, uma para a entrada igual a 0 e outra para a entrada igual a 1. Se, por outro lado, fossem definidas duas entradas, então haveria quatro possíveis combinações dos valores de entrada (00, 01, 10 e 11), pois $2^2 = 4$ e a tabela verdade possuiria quatro linhas e assim por diante.


Entradas
E_1
E_2
Porta lógica
X
R
Saída

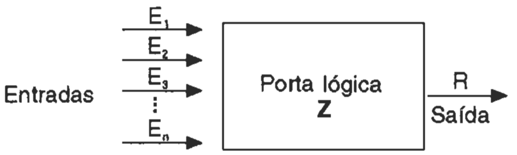
E_1 operação lógica X $E_2 = R$

Operação lógica X


Entrada
E
Porta lógica
Y
R
Saída

Operação lógica Y E = R

Operação lógica Y


Entradas
E_1
E_2
E_3
⋮
E_n
Porta lógica
Z
R
Saída

E_1 operação lógica Z E_2 operação lógica Z E_3
E_3 operação lógica Z . . . operação lógica Z $E_n = R$

Operação lógica Z

$E, E_1, E_2, E_3 \ldots E_n$ ⟶ Valores lógicos, 0 ou 1

Operações lógicas X, Y, Z ⟶ Operações lógicas ou portas lógicas

R ⟶ Resultado de uma operação lógica, 0 ou 1

Figura 4.2 Exemplo de configuração de operações lógicas.

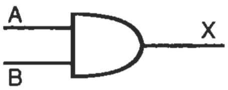
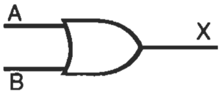
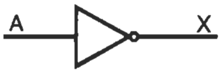
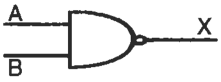
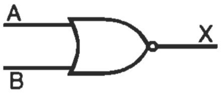
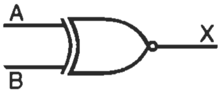
Porta lógica	Símbolo matemático	Símbolo gráfico
AND	· $X = A \cdot B$	
OR	+ $X = A + B$	
NOT	- $X = \overline{A}$	
NAND	$X = \overline{A \cdot B}$	
NOR	$X = \overline{A + B}$	
XOR	$\oplus$ $X = A \oplus B$	

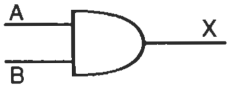
Figura 4.3 Símbolos gráficos e matemáticos de portas lógicas.

De um modo geral, a tabela verdade de uma dada operação lógica possui 2^n linhas ou combinações de valores de entrada, sendo *n* igual à quantidade de elementos de entrada.

4.2.1 Operação Lógica ou Porta AND (E)

A porta AND é definida como o elemento (ou operação lógica) que produz um resultado verdade (V = 1) na saída, *se e somente se todas as entradas forem verdade.* Essa definição pode ser expressa pela tabela verdade e símbolos mostrados na Fig. 4.4.

Uma porta lógica AND pode ter várias utilidades na fabricação de um sistema digital, algumas das quais são mostradas neste livro em diversas oportunidades. Entre elas, uma das mais importantes pode ser a de ativação de uma linha de dados para movimentar bits de um registrador (ou células) para outro.



$X = A \cdot B$ ou $X = AB$

Entrada		Saída
A	B	X = AB
0	0	0
0	1	0
1	0	0
1	1	1

Figura 4.4 Porta lógica AND (E).

A Fig. 4.5 mostra um exemplo de aplicação do circuito lógico (ou porta) AND como elemento de controle em transferências de dados. Para cada bit do registrador, um sinal da unidade de controle (UC) serve de entrada, juntamente com o sinal correspondente ao bit do registrador de origem (registrador A na Fig. 4.5); quando o sinal da unidade de controle for igual a 1 (pulso elétrico de intensidade e codificação correspondente ao bit 1) — *verdade* —, a combinação dos sinais de entrada produz na saída um valor, sempre igual ao do bit do registrador de entrada, o qual será armazenado no registrador de destino (registrador B na Fig. 4.5). Com isso, obteve-se a transferência dos bits do registrador de origem para o registrador de destino durante o período em que a linha da UC esteve com o bit 1 ativo. No item 4.4 serão abordadas outras aplicações de utilização de portas lógicas.

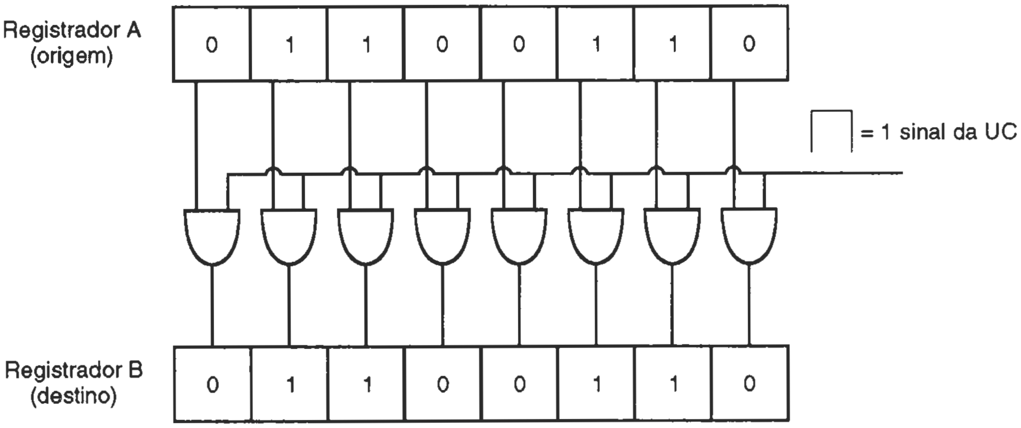


Figura 4.5 Exemplo de utilização de porta AND na movimentação de dados de um registrador para outro.

A combinação dos dois valores de entrada que estão exemplificados na Fig. 4.5 e que produziram resultados de acordo com a tabela verdade apresentada na Fig. 4.4 constitui, na realidade, uma operação lógica AND, porque foi realizada utilizando-se o operador lógico AND.

Operações lógicas AND podem ser realizadas para satisfazer um determinado requisito de hardware, como veremos no item 4.3 (Expressões Lógicas — Aplicações de Portas), ou para atender a uma especificação de um programador em um programa. Para tanto, a maioria dos processadores possui uma instrução de máquina AND em seu conjunto de instruções, bem como muitas linguagens de programação de alto nível implementam essa função para, como já mencionamos, atender a determinadas condições de programa.

Por exemplo, a lógica de um determinado programa pode estabelecer:

- Ler X, Y e Z
- T = X + Y
- R = Z + X
- SE (T > 6 E(AND) R < 10)
    - ENTÃO IMPRIMIR T
    - SENÃO IMPRIMIR R

O trecho de programa exemplificado é ridículo, mas exprime um exemplo do uso de AND na composição de uma condição a ser satisfeita (ser VERDADE) para que uma determinada ação seja realizada ou não.

No exemplo verificamos que o valor de T somente será impresso se ambas as condições forem verdadeiras: T > 6 e R < 10. O operador AND uniu ambas as afirmações.

Operações lógicas AND também podem ser realizadas com valores constituídos de vários algarismos (a UAL — Unidade Aritmética e Lógica realiza tal tipo de operação).

Exemplo 4.1

Seja A = 1 e B = 0. Calcular X = A · B (A **and** B).

Solução

Analisando a tabela verdade da Fig. 4.4 verificamos que:

X = 0, pois 1 **and** 0 = 0

Exemplo 4.2

Seja A = 0110 e B = 1101. Calcular X = A · B (A **and** B).

Solução

Pela tabela verdade da Fig. 4.4 teremos:

			A	B	X = A · B
0110	←	A	0	1	0
and 1101	←	B	1	1	1
			1	0	0
0100	←	X	0	1	0

Resultado: X = 0100

Exemplo 4.3

Seja A = 0101, B = 0011 e C = 1111. Calcular X = A · B · C (A **and** B **and** C).

Solução

O resultado é obtido através da realização das operações em duas etapas. Na primeira parte, calcula-se A · B (T = A **and** B) e, em seguida, o resultado parcial obtido (T) é combinado com C em outra operação lógica AND (T **and** C), sempre utilizando as combinações de entrada e os resultados definidos na tabela verdade da Fig. 4.4.

	A	B	T = A · B
0101 ← A	0	0	0
and 0011 ← B	1	0	0
	0	1	0
0001 ← T	1	1	1

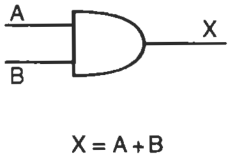
Resultado parcial: T = 0001

	T	C	X = T · C
0001 ← T	0	1	0
and 1111 ← C	0	1	0
	0	1	0
0001 ← X	1	1	1

Resultado: X = 0001

4.2.2 Operação Lógica ou Porta OR (OU)

A porta OR é definida para produzir um resultado verdade (V = 1) na sua saída, *se pelo menos uma das entradas for verdade*. Esta definição pode ser expressa pela tabela verdade e símbolos mostrados na Fig. 4.6.



Entrada		Saída
A	B	X = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Figura 4.6 Porta lógica OR (OU).

Operações lógicas OR também são largamente utilizadas em lógica digital (ver item 4.3) ou na definição de condições em comandos de decisão de certas linguagens de programação. No exemplo de trecho de programa mostrado no item anterior, pode-se modificar a condição do SE, tornando-a mais flexível:

- SE (T > 6 **OU** (OR) R < 10)
  - ENTÃO IMPRIMIR T
  - SENÃO IMPRIMIR R

Nesse caso, para T ser impresso, basta que uma das duas condições seja verdadeira (não *ambas*, como determina a operação AND), basta **ou** T > 6 **ou** R < 10. Não importa, inclusive, que ambas sejam verdadeiras, embora baste que apenas uma delas o seja.

Vejamos alguns exemplos de operações OR:

Exemplo 4.4

Seja A = 1 e B = 0. Calcular X = A + B (A **or** B).

Solução

Pela tabela verdade da Fig. 4.6, verificamos que:

X = 1, porque 1 **or** 0 = 1

Exemplo 4.5

Seja A = 0110 e B = 1110. Calcular X = A + B (A **or** B)

Pela tabela verdade da Fig. 4.6, teremos:

0110 ← A
or 1110 ← B
1110 ← T

A	B	T = A + B
0	1	1
1	1	1
1	1	1
0	0	0

Resultado: X = 1110

Exemplo 4.6

Seja A = 1100, B = 1111 e C = 0001. Calcular X = A + B + C (A **or** B **or** C).

Solução

O cálculo, como no Exemplo 4.3, também é realizado em duas etapas, utilizando-se a tabela verdade da Fig. 4.6. Na primeira parte, calcula-se A + B (T = A **or** B) e, em seguida, o resultado parcial obtido (T) é combinado com C em outra operação lógica OR (T **or** C), sempre utilizando as combinações de entrada e os resultados definidos na tabela verdade da Fig. 4.6.

1100 ← A
or 1111 ← B
1111 ← T

A	B	T = A + B
1	1	1
1	1	1
0	1	1
0	1	1

Resultado parcial: T = 1111

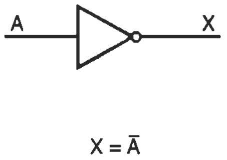
1111 ← T
or 0001 ← C
1111 ← X

T	C	X = T + C
1	0	1
1	0	1
1	0	1
1	1	1

Resultado: X = 1111

4.2.3 Operação Lógica NOT (Inversor)

A operação lógica NOT é também chamada de *inversor* ou *função complemento*. Ela inverte o valor de um sinal binário colocado em sua entrada, produzindo na saída o valor oposto. É um circuito lógico que requer apenas um valor na entrada (um outro circuito lógico — *buffer* — também requer apenas um valor de entrada, como será mostrado no item 4.4). A Fig. 4.7 mostra um circuito inversor, bem como os símbolos utilizados e a respectiva tabela verdade.



Entrada	Saída
A	$X = \bar{A}$
0	1
1	0

Figura 4.7 Porta lógica NOT (inversor ou inverter).

É interessante observar (ver Fig. 4.8) que a conexão de dois circuitos inversores em série produz, na saída, um resultado de valor igual ao da entrada. Ou seja, um duplo complemento restaura o valor original.

Em termos de representação gráfica do operador NOT, é possível também encontrar na literatura o apóstrofo (') para representar um circuito NOT, como exemplificado a seguir:

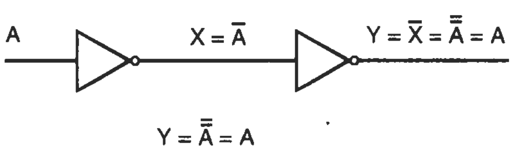
NOT A = A'

No entanto, adotaremos a barra em cima do valor de entrada

NOT A = $\bar{A}$

também mostrado na Fig. 4.7, simbologia definida pela ANSI (American National Standards Institute).

Uma das aplicações mais comuns do circuito inversor ou NOT é justamente em operações aritméticas em ponto fixo, quando se usa aritmética de complemento (complemento a 1 ou complemento a 2), conforme será mostrado no Cap. 7.



A = 1, então:

$X = \bar{A} = 0$, logo:

$Y = \bar{X} = 1 = A$

Figura 4.8 Exemplo de um duplo inversor, que restaura o valor de entrada.

A seguir, apresentaremos alguns exemplos de operações com circuitos inversores.

Exemplo 4.7

Seja A = 0. Calcular $X = \bar{A}$.

Solução

Utilizando a tabela verdade da Fig. 4.7, temos X = 1 porque $\bar{0} = 1$.

Exemplo 4.8

Seja A = 10011. Calcular $X = \overline{A}$.

Solução

Utilizando a tabela verdade da Fig. 4.7, obtemos o valor de X invertendo o valor de cada algarismo.

10011 ← A

01100 ← inverso de A, bit a bit

$X = 01100 = \overline{A}$

Exemplo 4.9

Seja A = 10010 e B = 11110. Calcular $X = \overline{A \cdot B}$.

Solução

Trata-se da realização de duas operações lógicas em seqüência. Primeiro, realiza-se a operação lógica AND e, em seguida, obtém-se o inverso do resultado.

Pela tabela verdade da Fig. 4.4, temos:

	A	B	T = A · B
10010 ← A	1	1	1
and 11110 ← B	0	1	0
	0	1	0
10010 ← T	1	1	1
	0	0	0

Resultado parcial: T = 10010

Inverte-se T, usando a tabela verdade da Fig. 4.7:

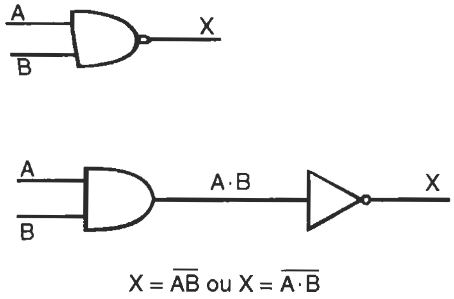
10010 ← T → A · B.

01101 ← $\overline{T}$ → $\overline{A \cdot B}$.

Resultado: $X = \overline{T} = \overline{A \cdot B} = 01101$

4.2.4 Operação Lógica NAND — NOT AND

A operação lógica ou porta NAND é definida como o complemento da porta AND, isto é, a saída de um circuito lógico NAND (que é o mesmo resultado da operação lógica NAND) é obtida ao se aplicar a regra da operação lógica AND e inverter o resultado. São, então, dois passos, conforme pode ser visto pelos símbolos mostrados na Fig. 4.9.



$X = \overline{AB}$ ou $X = \overline{A \cdot B}$

Entrada		Saída
A	B	$X = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

Figura 4.9 Porta lógica NAND ou NOT AND.

Deve ser observado que o valor encontrado em:

$\overline{A \cdot B}$

não é o mesmo valor obtido em:

$\overline{A} \cdot \overline{B}$

Se não, vejamos:

Exemplo 4.10

Seja A = 1 e B = 0. Calcular: a) $X = \overline{A \cdot B}$ e b) $Y = \overline{A} \cdot \overline{B}$.

Solução

Adotando a tabela verdade da Fig. 4.4 para a operação AND e a tabela verdade da Fig. 4.7 para a inversão (NOT), teremos:

a) $A \cdot B = 1 . 0 = 0$ e $\overline{A \cdot B} = \overline{0} = 1$

b) Para: $X = \overline{A} \cdot \overline{B} = 1$

Se A = 1, então: $\overline{A} = 0$ (porque NOT 1 = 0), e se B = 0, então: $\overline{B} = 1$ (porque NOT 0 = 1)

Então: $X = \overline{A} \cdot \overline{B} = 0 \cdot 1 = 0$

O circuito lógico NAND realiza a operação de inversão do AND; logo, é a operação usada no Exemplo 4.10 (a).

A porta NAND produzirá uma saída falsa (ver tabela da Fig. 4.8) *se e somente se todas as entradas forem verdade.* Do contrário, a saída será verdade se, pelo menos, uma entrada for falsa.

É muito comum encontrar esta porta em complexos circuitos lógicos, visto que é possível simplificar a fabricação de circuitos lógicos e reduzir a quantidade de componentes eletrônicos se usarmos apenas circuitos NAND. No item 4.3 veremos algumas dessas aplicações, mas podemos exemplificar desde já mostrando, na Fig. 4.10, alguns usos de circuitos constituídos apenas de portas NAND.

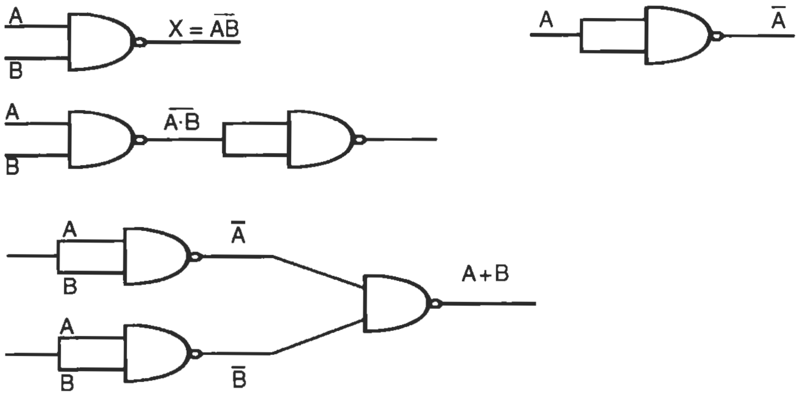


Figura 4.10 Usos da porta NAND.

Exemplo 4.11

Seja A = 1 e B = 1. Calcular o valor de X em $X = \overline{A \cdot B}$.

Solução

Utilizando as tabelas verdade das Figs. 4.4 e 4.7, teremos:

$A = 1 \cdot B = 1$, ou $1 \cdot 1 = 1$

$X = \overline{1 \cdot 1} = \bar{1} = 0$

$X = 0$

Exemplo 4.12

Seja A = 10110 e B = 00011. Calcular o valor de X em $X = \overline{A \cdot B}$.

Solução

Utilizando as mesmas tabelas verdade do exemplo anterior (Figs. 4.4 e 4.7), calcularemos o valor de X em duas etapas: em primeiro lugar efetuaremos a operação AND de A e B, obtendo um resultado parcial $T = A \cdot B$. Em seguida, o valor do resultado parcial T será invertido, utilizando o operador NOT.

	A	B	$T = A \cdot B$
10110 ← A	1	0	0
and 00011 ← B	0	0	0
	1	0	0
10010 ← T	1	1	1
	0	1	0

Resultado parcial: T = 00010

Invertendo o resultado parcial T:

T = 00010 e $\overline{T}$ = 11101

Resultado final: $X = \overline{A \cdot B} = 11101$

Exemplo 4.13

Seja A = 11110, B = 01001 e C = 10000. Calcular: $X = \overline{A \cdot B \cdot C}$.

Solução

Utilizando as mesmas tabelas verdade (Figs. 4.4 e 4.7), realizam-se as operações em três etapas. Primeiro, $T_1 = A \cdot B$. A segunda etapa é $T_2 = T_1 . C$ e, finalmente, na terceira etapa, inverter o resultado parcial T_2.

	A	B	$T_1 = A \cdot B$
11110 ← A	1	0	0
and 01001 ← B	1	1	1
	1	0	0
01000 ← T_1	1	0	0
	0	1	0

1.ª etapa — Resultado parcial: T_1 = 01000

	T_1	C	$T_2 = T \cdot C$
01000 ← T_1	0	1	0
and 10000 ← C	1	0	0
	0	0	0
00000 ← T_2	0	0	0
	0	0	0

2.ª etapa — Resultado parcial: $T_2 = 00000$

3.ª etapa — Inversão de T_2: Se $T_2 = 0000$, então: $X = \overline{T_2} = 11111$

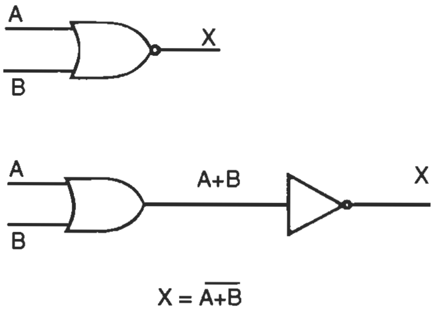
Resultado final: $X = \overline{A \cdot B \cdot C} = 11111$

4.2.5 Operação Lógica NOR — NOT OR

Assim como a porta NAND, a porta NOR é o complemento ou o inverso da porta OR. A saída de um circuito lógico NOR é obtida ao se efetuar a operação lógica OR sobre as entradas e inverter o resultado. Também esta operação lógica (NOR) é realizada em dois passos: primeiro a operação OR e, em seguida, a inversão (ou operação NOT) do resultado.

A Fig. 4.11 mostra os símbolos e a tabela verdade da operação lógica NOR.

A porta NOR apresentará uma saída verdade *se e somente se todas as entradas forem falsas*. Caso contrário, a saída será falsa (se, pelo menos, uma das entradas for verdade).



Entrada		Saída
A	B	$X = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Figura 4.11 Porta lógica NOR ou NOT OR.

Assim como verificamos com a porta NAND, também o valor encontrado como resultado na operação:

$\overline{A + B}$, que é a implementação da porta NOR,

por definição não é o mesmo valor obtido com a operação:

$\overline{A} + \overline{B}$

Exemplo 4.14

Seja A = 1 e B = 0. Calcular o valor de X e de Y nas expressões: a) $X = \overline{A + B}$ e b) $Y = \overline{A} + \overline{B}$.

Solução

Adotando a tabela verdade da Fig. 4.6 para a operação OR e a tabela verdade da Fig. 4.7 para a operação NOT, teremos:

a) $A + B = 1 + 0 = 1$ e $X = \overline{A + B} = \overline{1} = 0$

Então: $X = \overline{A + B} = 0$

b) Se $A = 1$, então: $\overline{A} = \overline{1} = 0$ e se $B = 0$, então: $\overline{B} = \overline{0} = 1$

Assim, $Y = \overline{A} + \overline{B} = 0 + 1 = 1$

É possível projetar circuitos lógicos mais complexos utilizando apenas portas NOR, de modo semelhante ao que já foi mostrado para o caso de portas NAND. A Fig. 4.12 mostra alguns exemplos somente com o emprego de circuitos lógicos NOR.

Em seguida, são apresentados alguns exemplos da aplicação de circuitos lógicos NOR em operações lógicas.

Exemplo 4.15

Seja $A = 0$ e $B = 1$. Calcular: $X = \overline{A+B}$.

Solução

O cálculo é efetuado em duas etapas, e utilizando as tabelas verdade das Figs. 4.6 e 4.7. Na primeira, realiza-se a operação OR (Tabela 4.6) e, em seguida, inverte-se o resultado parcial anterior (operação NOT Tabela 4.7).

$A = 0 + B = 1$, ou $0 + 1 = 1$

Inversão do resultado: $\overline{1} = 0$

Resultado: $X = \overline{A + B} = \overline{1} = 0$

Exemplo 4.16

Seja A = 10001 e B = 01010. Calcular: $X = \overline{A+B}$.

Solução

Trata-se da realização de duas operações lógicas em seqüência. Primeiro, a operação lógica OR, obtendo-se o resultado parcial T. Em seguida, obtém-se o valor de X, invertendo-se (operação NOT) o valor de T.

Pela tabela verdade da Fig. 4.6, temos:

	A	B	T = A + B
10001 ← A	1	0	1
or 01010 ← B	0	1	1
	0	0	0
11011 ← T	0	1	1
	1	0	1

Resultado parcial: T = 11011

Inverte-se T, usando a tabela verdade da Fig. 4.7:

11011 ← T → A + B

00100 ← $\overline{T}$ → $\overline{A + B}$

Resultado: $X = \overline{A + B} = 00100$

Exemplo 4.17

Seja A = 11110, B = 10011 e C = 00100. Calcular: $X = \overline{A + B + C}$.

Solução

Utilizando as mesmas tabelas verdade (Figs. 4.6 e 4.7), realizam-se as operações em três etapas. Na primeira etapa, $T_1 = A + B$; na segunda etapa, $T_2 = T_1 + C$ e, finalmente, na terceira etapa, inverter o resultado parcial T_2.

	A	B	$T_1 = A + B$
11110 ← A	1	1	1
or 10011 ← B	1	0	1
	1	0	1
11111 ← T_1	1	1	1
	0	1	1

1.ª etapa: Resultado parcial: $T_1 = 11111$

	T_1	C	$T_2 = T + C$
11111 ← T_1	1	0	1
or 00100 ← C	1	0	1
	1	1	1
11111 ← T_2	1	0	1
	1	0	1

2.ª etapa: Resultado parcial: $T_2 = 11111$

3.ª etapa: Inversão de T_2: Se $T_2 = 11111$, então: $X = \overline{T_2} = 00000$

Resultado final: $X = \overline{A + B + C} = 00000$

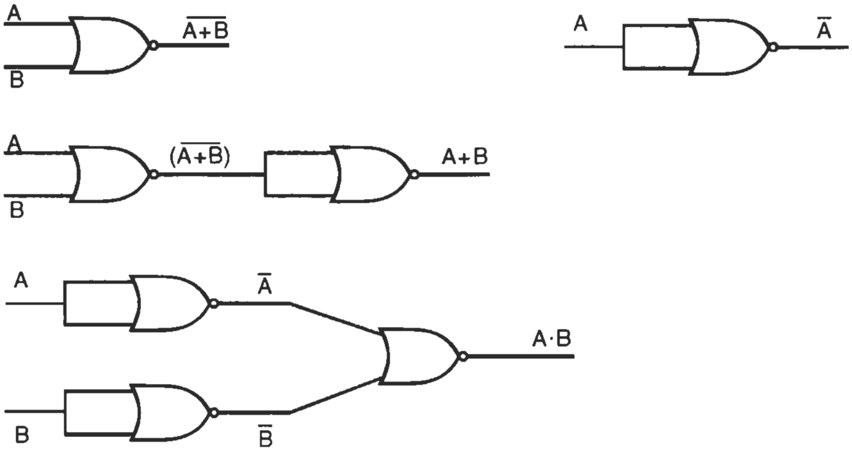


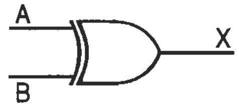
Figura 4.12 Exemplo de uso de circuitos lógicos NOR.

4.2.6 Operação Lógica XOR — EXCLUSIVE OR

A operação lógica XOR, abreviação do termo EXCLUSIVE OR, pode ser considerada um caso particular da função OR, ou seja, sua definição: "a saída será verdade se *exclusivamente* uma ou outra entrada for verdade". Isto só se aplica, é claro, se houver apenas 2 entradas e o termo *exclusivamente* é crucial. Não podem ambas as entradas ser verdade e é esta a diferença para os resultados da porta OR (ver Tabela 4.6).

A Fig. 4.13 mostra os símbolos e a tabela verdade para a operação lógica XOR, e realmente podemos verificar que os resultados mostrados na tabela são iguais aos da operação OR, exceto quanto ao resultado de $1 \oplus 1$. Isto por causa do termo *exclusivo*. No caso da operação lógica OR, não há exclusividade de que uma ou outra entrada sejam verdadeiras, mas no que se refere à operação XOR, sim, isto é, na operação XOR é necessário que uma ou outra entrada seja verdade; se forem ambas verdade o resultado é falso.

O XOR é um elemento lógico bastante versátil, aparecendo em diversos locais e com diferentes utilidades em circuitos digitais. A saída (resultado) de uma operação XOR será *verdade se os valores de suas entradas forem diferentes* e será *falsa se os valores das entradas forem iguais*. Esta definição se aplica melhor do que a que mencionamos anteriormente, porque é abrangente para qualquer quantidade de linhas de entrada.



$X = A \oplus B$

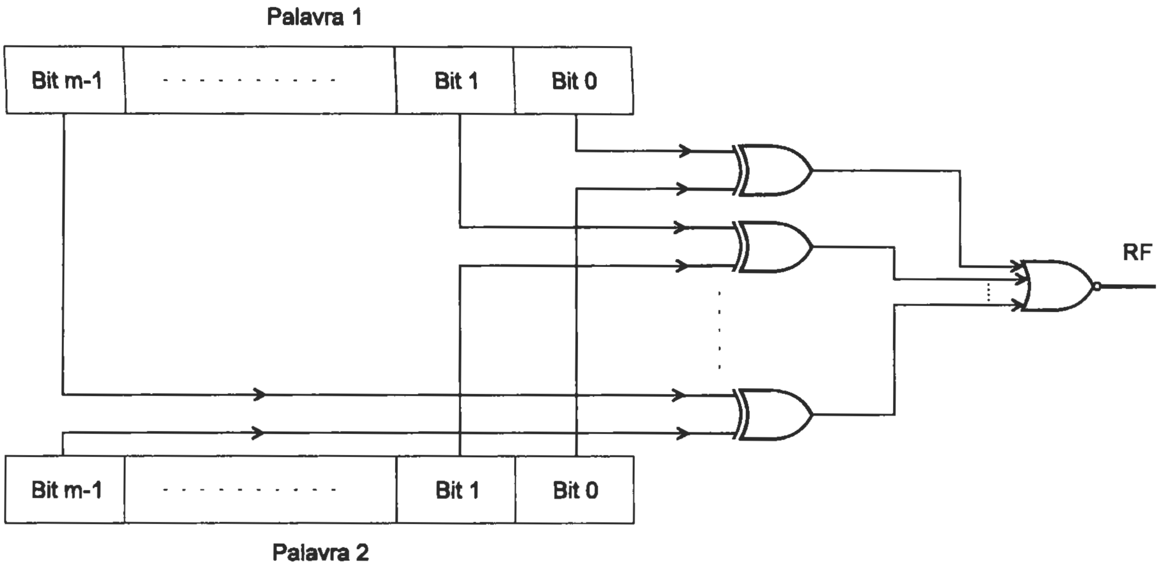
Entrada		Saída
A	B	$X = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Figura 4.13 Porta lógica XOR (exclusive or = ou exclusivo).

Além disso, esta particularidade dos circuitos lógicos XOR permite que se fabrique um testador de igualdade entre valores, por exemplo, para testar, de modo rápido, se duas palavras de dados são iguais. Se as duas palavras forem iguais (ver exemplo na Fig. 4.14), as saídas do circuito XOR serão todas FALSAS. A figura mostra o exemplo completo do teste, que inclui um circuito lógico NOR para receber, como entrada, a saída de todos os circuitos XOR (um para o teste de cada bit das duas palavras) e produzir uma única saída.

Se as palavras forem iguais, isso significa que o bit 0 da palavra 1 e o bit 0 da palavra 2 têm valores iguais e que cada par de bits até o bit (m-l) das duas palavras também são iguais. Isto garante que, se as entradas de cada circuito lógico XOR forem iguais, a sua saída será FALSA. Se todas as entradas do circuito lógico NOR forem FALSAS, então sua saída será VERDADE e este valor (bit 1) será armazenado em um registrador especial de estado da UCP (denominado Flag em algumas UCP ou Código de Condição em outras, como veremos no Cap. 6).

O operador lógico XOR pode servir, com este mesmo tipo de teste, para verificar se os sinais de dois valores são iguais ou não (no caso, por exemplo, de valores representados em sinal e magnitude, como mostrado no item 7.5.1).



$$\begin{array}{rl} \text{Palavra 1} = 0\,0\,1\,0\,0\,1\,1 & \\ & \text{XOR} \\ \text{Palavra 2} = 1\,0\,1\,0\,0\,1\,1 & \\ \hline 1\,0\,0\,0\,0\,0\,0 & \text{T} \\ 7\,6\,5\,4\,3\,2\,1 & \end{array}$$

$\overline{(T_1+T_2=T_3+T_4+T_5+T_6+T_7)} = 0$ (ver tabela verdade da Fig.4.11)

$1+0+0+0+0+0+0=1$ invertendo, resulta em 0.

Figura 4.14 Exemplo de emprego de circuitos XOR para teste de igualdade entre duas palavras de dados.

Vamos, em seguida, apresentar alguns exemplos de operações lógicas XOR.

Exemplo 4.18

Seja A = 0 e B = 1. Calcular: $X = A \oplus B$.

Solução

Utilizando os resultados da tabela verdade da Fig. 4.13, teremos:

$A = 0 \oplus B = 1$ ou $0 \oplus 1 = 1$

Resultado: $X = A \oplus B = 1$

Exemplo 4.19

Seja A = 11001 e B = 11110. Calcular: $X = A \oplus B$.

Solução

Adotando a tabela verdade da Fig. 4.13, teremos:

	A	B	$T = A \oplus B$
11001 ← A	1	1	0
xor 11110 ← B	1	1	0
	0	1	1
00111 ← T	0	1	1
	1	0	1

Resultado: $X = A \oplus B = 00111$

Exemplo 4.20

Seja A = 11001, B = 10011 e C = 11100. Calcular: $X = A \oplus B \oplus C$.

Solução

Utilizando a mesma tabela verdade (Fig. 4.13), realiza-se a operação em duas etapas. Na primeira etapa, $T = A \oplus B$ e na segunda etapa, $X = T \oplus C$.

	A	B	$T = A \oplus B$
11001 ← A	1	1	0
xor 10011 ← B	1	0	1
	0	0	0
01010 ← T	0	1	1
	1	1	0

1.ª etapa: Resultado parcial: T = 01010

	T	C	$X = T \oplus C$
01010 ← T	0	1	1
xor 11100 ← C	1	1	0
	0	1	1
10110 ← X	1	0	1
	0	0	0

2.ª etapa: Resultado final: X = 10110

$X = A \oplus B \oplus C = 10110$

4.3 EXPRESSÕES LÓGICAS — APLICAÇÕES DE PORTAS

Uma expressão lógica ou função lógica pode ser definida como uma expressão algébrica formada por variáveis lógicas (binárias), por símbolos representativos de uma operação lógica (+, . , ⊕ etc.), por parênteses (às vezes) e por um sinal de igualdade ("=").

Por exemplo:

$$F = X + \overline{Y} \cdot Z$$

Neste exemplo, F, que é uma função lógica, é representada pela expressão lógica mostrada. E como função lógica, somente poderá assumir os valores 0 ou 1, dependendo do valor das variáveis X, Y e Z. F será igual a

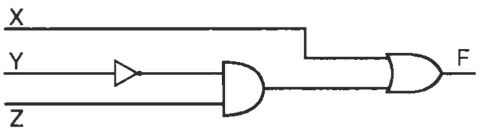
1 (será VERDADE) se X for igual a 1 OU (OR) se ambos $\overline{Y}$ E (AND) Z forem iguais a 1 (para isso, Y = 0). É possível representar a função F de duas maneiras:

- pela expressão algébrica ou expressão lógica acima mostrada; e
- por um diagrama interligando os símbolos gráficos correspondentes às operações lógicas.

E o valor do resultado de uma expressão lógica pode ser obtido por uma tabela verdade construída com todas as possibilidades de entrada e as correspondentes saídas. A Fig. 4.15 mostra a função F, utilizada como exemplo, representada das duas maneiras citadas, sendo que no exemplo (a) apresenta-se a expressão lógica; no exemplo (b) é mostrado o diagrama lógico da função F e no exemplo (c) a correspondente tabela verdade. Esta, por se tratar de três entradas (X, Y e Z), possui $2^3 = 8$ combinações possíveis.

$F = X + \overline{Y} \cdot Z$

(a) Expressão lógica da função F



(b) Diagrama lógico da função F

Entrada			Saída
X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(c) Tabela verdade da função F

Figura 4.15 Representação da função F = X or (not Y) and Z.

Uma boa maneira de avaliar expressões lógicas consiste em construir a tabela verdade final da expressão através da obtenção de todas as possibilidades de forma progressiva, operando por operando, até chegarmos ao valor da função para cada possibilidade.

Vejamos o mesmo exemplo anterior da função $F = X + \overline{Y} + Z$; vamos obter a tabela verdade final, igual à da Fig. 4.15(c), passo a passo, através de sucessivas tabelas verdade, como mostrado na Fig. 4.16.

Em primeiro lugar, verificamos que a tabela deve ter 8 entradas, pois temos 3 variáveis e sabemos que $2^3 = 8$, conforme está mostrado nos valores de X, Y, Z da Fig. 4.16(a). Como Y é invertido (complementado) na expressão, então cria-se outra coluna com o valor de $\overline{Y}$, como mostrado na Fig. 4.16(b). Como desejamos obter o valor de $\overline{Y} \cdot Z$, então acrescenta-se outra coluna à tabela com o resultado de $\overline{Y} \cdot Z$ para cada uma das 8 entradas possíveis, o que está mostrado na Fig. 4.16(c). E, finalmente, completa-se o cálculo da expressão, incluindo a coluna com o resultado de $X + \overline{Y} \cdot Z$ para cada uma das 8 possibilidades de entrada, conforme mostrado na Fig. 4.16(d), que é igual à tabela verdade apresentada inicialmente na Fig. 4.15(c).

4.3.1 Cálculos com Expressões Lógicas

Assim como podemos obter todos os possíveis resultados de uma expressão lógica para cada um dos valores de entrada, componentes da expressão (através da construção progressiva da tabela verdade), também podemos obter o valor da expressão para um valor específico de cada uma das entradas (usar apenas uma linha da tabela verdade).

Expressões lógicas podem ou não conter parênteses; quando contêm, eles têm a mesma prioridade para o cálculo que parênteses de equações algébricas comuns.

X	Y	Z
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

(a) Possibilidade de X, Y e Z

X	Y	$\overline{Y}$	Z
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1

(b) Inclusão de $\overline{Y}$

X	Y	Z	$\overline{Y}$	$\overline{Y}\cdot Z$
0	0	0	1	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

(c) Avaliação de $\overline{Y}\cdot Z$

X	Y	Z	$\overline{Y}$	$\overline{Y}\cdot Z$	$X+\overline{Y}\cdot Z$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

(d) Avaliação de $X+\overline{Y}\cdot Z$

Figura 4.16 Avaliação da expressão $F = X + \overline{Y} \cdot Z$ por tabelas verdade.

A prioridade de cálculo de uma operação AND é maior do que a do cálculo de uma operação OR, semelhante ao que temos na aritmética comum, em que a multiplicação (representada por um ponto (·), assim como o AND) tem prioridade sobre a adição (representada por um sinal +, igual ao da operação lógica OR).

Assim, na expressão:

$$X + \overline{Y} \cdot Z$$

o primeiro cálculo a ser realizado é:

$\overline{Y} \cdot Z$ e não $X + \overline{Y}$

como se a expressão tivesse parênteses:

$$X + (\overline{Y} \cdot Z)$$

O que é desnecessário, dado o conhecimento da prioridade de cálculo, isto é, AND (·) tem prioridade sobre OR (+).

Vamos mostrar alguns exemplos do cálculo do valor de expressões lógicas, isto é, como obter o resultado de uma expressão lógica (que pode ser apenas = 0 ou = 1) para um dado valor (= 0 ou = 1) de cada variável componente da expressão. Como se trata de um valor específico de cada variável, a construção de cada tabela verdade seria tediosa e demorada. Neste caso, é mais simples realizar os cálculos passo a passo.

Exemplo 4.21

Seja A = 1, B = 0, C = 1, D = 1. Calcular: $X = A + \overline{B \cdot C} \oplus D$.

Solução

Adotando o esquema de prioridade, então o valor de X será obtido com a realização de quatro etapas.

1) Realizar a operação AND (maior prioridade, além de ter uma inversão determinada sobre a operação). Assim, trata-se de calcular $B \cdot C = T_1$.
2) Inverter o resultado parcial T_1.
3) Realizar a operação OR (as operações OR e XOR têm mesma prioridade, optando-se pela que está primeiro à esquerda). Assim, calcula-se: $T_2 = A + T_1$.
4) Realizar a operação XOR, calculando-se $X = T_2 \oplus D$.

Desse modo, vamos efetuar as etapas indicadas:

1) $0 \cdot 1 = 0 = T_1$ 2) $\overline{0} = 1$

3) $1 + 1 = 1 = T_2$ 4) $1 \oplus 1 = 0 = X$

Resultado: $X = 0$

Exemplo 4.22

Seja $A = 0$, $B = 0$, $C = 1$, $D = 1$. Calcular: $X = \overline{(A+\overline{B}\oplus D)}+(\overline{C}\cdot B) \oplus A$

Solução

Adotando a seqüência de etapas semelhantes à do exemplo anterior e considerando as prioridades envolvidas (a primeira prioridade é solucionar os parênteses, e dentro ou fora destes a prioridade é da operação AND sobre as demais, exceto se houver inversão (NOT)). Assim, temos:

1) Calcular o parêntese mais à esquerda; dentro deste parêntese, efetuar primeiro a inversão do valor de B: $B = 0$ e not $B = 1$.

2) Ainda dentro do parêntese, efetuar A + not B ou $0 + 1 = 1 = T_1$.

3) Em seguida, encerra-se o cálculo do interior do parêntese efetuando $T_1 \oplus D = T_2$ e invertendo o resultado total do parêntese:

   a) $1 \oplus 1 = 0 = T_2$

   b) $T_2 = 0$ e $\overline{T_2} = 1$

4) Então, calcula-se o outro parêntese, primeiro invertendo o valor de C (NOT), depois efetuando a operação AND daquele resultado com a variável B. O valor final é, temporariamente, T_3.

   a) $C = 1$ e not $C = 0$

   b) $0 \cdot 0 = 0 = T_3$

5) Finalmente calcula-se a operação OR do resultado final do primeiro parêntese (T_2) com o do outro parêntese (T_3), para concluir com a operação XOR com A.

   a) $1 + 0 = 1$

   b) $X = 1 \oplus 0 = 1$

   Resultado: $X = 1$

Exemplo 4.23

É possível, também realizar operações lógicas com palavras de dados, isto é, com variáveis de múltiplos bits.

Seja A = 1001, B = 0010, C = 1110, D = 1111. Calcular o valor de X na seguinte expressão lógica:

$X = A \oplus (\overline{B \cdot C} + D) + (B \oplus \overline{D})$

Solução

Neste exemplo segue-se o mesmo método, execução por etapas, dos exemplos anteriores, com a diferença de que são 4 algarismos binários em vez de um apenas.
Considerando as prioridades já definidas anteriormente, temos:

1) Executar a operação AND de B e C, obtendo resultado parcial T_1.
2) Inverter o valor de T_1 (not T_1).
3) Executar a operação OR de not T_1 com D, atualizando um novo resultado parcial T_1, que é a solução do primeiro parêntese.
4) Inverter o valor de D no segundo parêntese.
5) Executar a operação XOR de B com o inverso de D, obtendo o resultado parcial T_2, que é a solução do segundo parêntese.
6) Executar a operação XOR de A com T_1, obtendo um valor temporário para X.
7) Executar a operação OR de X com T_2, obtendo o resultado final de X.

Executando as etapas aqui indicadas, teremos:
Etapa 1: $T_1 = B \cdot C$

0010 ← B
and 1110 ← C
0010 ← T_1

B	C	$T_1 = B \cdot C$
0	1	0
0	1	0
1	1	1
0	0	0

Resultado parcial: $T_1 = 0010$

Etapa 2: $T_1 = \text{not } T_1$

$T_1 = 0010$ e not $T_1 = 1101$

Resultado parcial: novo $T_1 = 1101$

Etapa 3: $T_1 = T_1 + D$

1101 ← T_1
or 1111 ← D
1111 ← T_1

T_1	D	$T_1 = T_1 + D$
1	1	1
1	1	1
0	1	1
1	1	1

Resultado parcial: T_1 atualizado: $T_1 = 1111$

Etapa 4: not D

D = 1111 e not D = 0000

Etapa 5: $T_2 = B \oplus \text{not } D$

0010 ← B
xor 0000 ← not D
0010 ← T_2

B	not D	$T_2 = B \oplus \text{not } D$
0	0	0
0	0	0
1	0	1
0	0	0

Resultado parcial T_2: 0010

Etapa 6: $X = A \oplus T_1$

	A	T_1	$X = A \oplus T_1$
1001 ← A	1	1	0
xor 1111 ← T_1	0	1	1
	0	1	1
0110 ← X	1	1	0

Resultado parcial: $X = 0110$

Etapa 7: $X = X + T_2$

	X	T_2	$X = X + T_2$
0110 ← X	0	0	0
xor 0010 ← T_2	1	0	1
	1	1	1
1110 ← X	0	0	0 .

Resultado final: $X = 0110$

Exemplo 4.24

Solucionar a expressão apresentada no Exemplo 4.22, utilizando os mesmos valores das variáveis daquele exemplo. No entanto, executar as etapas através da composição completa de todas as tabelas verdade envolvidas nos cálculos.

Solução

Neste exemplo, idêntico ao Exemplo 4.22, vamos adotar um processo mais demorado ao obter as tabelas verdade passo a passo, e para todos os possíveis valores de resultado. De modo que teremos um resultado final (valor de X) para cada uma das 16 possibilidades (A, B, C, D são quatro entradas, haverá, portanto, $2^4 = 16$ saídas). Das 16 saídas, será escolhida aquela que corresponde aos valores de entrada de dados definidos no exemplo. Se as entradas fossem palavras com tamanho maior que um bit, então seria necessário construir tantos conjuntos de tabelas verdade quanto fosse a quantidade de bits da palavra.

Ao se comparar a quarta linha da tabela exemplo 4.24 — item 1), cujo valor final é igual a 1, com o resultado do Exemplo 4.22, que também é igual a 1, verifica-se serem iguais . E mais ainda, também se pode constatar que o resultado obtido no Exemplo 4.22 consumiu um tempo menor pela ausência da elaboração de tantas tabelas verdade.

Vamos construir as tabelas, uma para cada operação a ser efetuada, como definido a seguir:

1) Calcular o parêntese mais à esquerda; dentro deste parêntese, efetuar primeiro a inversão do valor de B: $B = 0$ e not $B = 1$.

2) Ainda dentro do parêntese, efetuar A + not B ou $0 + 1 = 1 = T_1$.

3) Em seguida, encerra-se o cálculo do interior do parêntese efetuando $T_1 \oplus D = T_2$ e invertendo o resultado total do parêntese:

    a) $1 \oplus 1 = 0 = T_2$.

    b) $T_2 = 0$ e $\overline{T_2} = 1$

4) Então, calcula-se o outro parêntese, primeiro invertendo o valor de C (NOT), depois efetuando a operação AND daquele resultado com a variável B. O valor final é, temporariamente, T_3.

    a) $C = 1$ e $\overline{C} = 0$

    b) $0 \cdot 0 = 0 = T_3$

5) Finalmente calcula-se a operação OR do resultado final do primeiro parêntese (T_2) com o do outro parêntese (T_3), para concluir com a operação XOR com A.
   a) not $T_2 = 1 + T_3 = 0 = 1 + 0 = 1$

   b) $X = 1 \oplus 0 = 1$

   Resultado: $X = 1$

A	B	C	D
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Ex. 4.24

Etapa 1: not B				
A	B	C	D	$\overline{B}$
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Ex. 4.24 — item 1

Etapa 2: T_1 = A + not B				
A	B	C	D	$(A + \overline{B})$
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Ex. 4.24 — item 2

Etapa 3a: $T_2 = T_1 \oplus D$				
A	B	C	D	$T_2 = (A + \overline{B}) \oplus D$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Ex. 4.24 — item 3a

Etapa 3b: not T_2					
A	B	C	D	T_2	$\overline{T_2}$
0	0	0	0	1	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	0	1

Exemplo 4.24 — item 3b

Etapa 4a: not C				
A	B	C	D	$\overline{C}$
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Exemplo 4.24 — item 4a

Etapa 4b: T_3 = not C · B					
A	B	C	D	$\overline{C}$	$T_3 = (\overline{C} \cdot B)$
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	0	0

Exemplo 4.24 — item 4b

Etapa 5a: X = not T_2 + T_3

A	B	C	D	$\overline{T}_2$	T_3	X = $\overline{T}_2 + T_3$
0	0	0	0	0	0	0
0	0	0	1	1	0	1
0	0	1	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	1
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	1	0	1
1	0	1	0	0	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	1	1	1	0	1

Exemplo 4.24

Etapa 5b: X = X ⊕ A

A	B	C	D	X	X = X ⊕ A
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	1	1	1
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	0	0
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	1	1	0

Exemplo 4.24 — item 1

a) Como são 4 variáveis, A, B, C e D, então temos 16 possibilidades, Tabela 4.24a.
b) Cada uma das etapas está implementada por tabela verdade correspondente.
c) Finalmente, verifica-se que o resultado da etapa 5b indica um valor para X, igual a 1 na 4.ª linha. Nesta linha temos o seguinte conjunto de 4 bits:

0011

correspondendo aos valores A = 0, B = 0, C = 1 e D = 1, conforme enunciado do exemplo. Este é o mesmo valor do Exercício 4.22.

4.4 UM POUCO MAIS DE DETALHE

4.4.1 Introdução

Neste item serão apresentados alguns detalhes sobre Álgebra Booleana e Lógica Digital, que poderão interessar ao leitor mais curioso ou que necessite obter informações adicionais sobre o assunto. O primeiro tópico a ser abordado refere-se à álgebra booleana, conceituando esta parte da matemática, suas regras fundamentais, identificando vantagens de seu emprego na análise e no projeto de circuitos digitais. Em seguida, trataremos de circuitos combinatórios e de sua importância no projeto de computadores. Finalmente, serão apresentados alguns aspectos de lógica digital, descrevendo componentes básicos encontrados nos computadores digitais, como decodificadores e flip-flops.

4.4.2 Noções de Álgebra Booleana

Álgebra booleana é uma área da matemática que trata de regras e elementos de lógica. O nome "booleana" é uma retribuição da comunidade científica ao matemático inglês George Boole (1815-1864), que desenvolveu uma análise matemática sobre a Lógica. Em 1854, ele publicou o famoso livro "An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities" (Uma investigação das leis do pensamento nas quais estão alicerçadas teorias matemáticas de lógica e probabilidade), no qual ele propôs os princípios básicos dessa álgebra.

Talvez a álgebra booleana se tornasse apenas uma ferramenta de filosofia, se não tivesse ocorrido o desenvolvimento tão acentuado da eletrônica neste século e a grande utilização da lógica digital nesse processo.

Em 1938, Claude Shannon, no MIT (Massachusetts Institute of Technology), utilizou os conceitos dessa álgebra para o projeto de *circuitos de chaveamento* (*switching circuits*) que usavam relés. Tais circuitos estavam ligados a centrais de comutação telefônica e, mais tarde, a computadores digitais. Atualmente, essa álgebra é largamente empregada no projeto de circuitos digitais, para:

- *análise* — é um método prático e econômico de descrever as funções de um circuito digital e, conseqüentemente, seu funcionamento; e
- *projeto* — ao identificar a função a ser realizada por um circuito, a álgebra booleana pode ser aplicada para simplificar sua descrição e, assim, também sua implementação.

O projeto de elementos digitais está relacionado com a conversão de idéias em hardware real, e os elementos encontrados na álgebra booleana permitem que uma idéia, uma afirmação, possa ser expressa matematicamente. Permitem também que a expressão resultante da formulação matemática da idéia possa ser simplificada e, finalmente, convertida no mundo real do hardware de portas lógicas (*gates*) e outros elementos digitais.

Assim como na álgebra comum, a álgebra booleana trata de variáveis e de operações a serem realizadas com essas variáveis. A diferença é que, no que se refere à álgebra booleana, as variáveis usadas são binárias, tendo apenas dois valores possíveis, VERDADE — V (equivalente ao bit 1) e FALSO — F (equivalente ao bit 0). Da mesma forma que o bit só possui dois valores, também um relé ou uma porta lógica pode estar em uma de duas possíveis posições: ou na posição ABERTO (V = bit 1) ou na posição FECHADO (F = bit 0).

Por exemplo, observemos a expressão de uma idéia:

"*A lâmpada acenderá se o sinal A estiver presente.*"

Esta afirmação implica a dependência de uma ação sobre outra. Se criássemos o indicador L para o fato de a lâmpada acender e o indicador A para o sinal A, poderíamos estabelecer uma equação simbólica para expressar esta afirmação (ver Fig. 4.17(a)).

$L = A$ (à direita) e $L \neq A$ (à esquerda)

Poderíamos também modificar a expressão para indicar:

"*A lâmpada acenderá se a chave A* **E** *(AND) a chave B estiverem fechadas.*"

Neste caso, a expressão será diferente, aparecendo a conexão E (AND), conforme exemplo de um circuito em série, mostrado na Fig. 4.17(b).

L = A AND B

Ou ainda a expressão:

"*A lâmpada somente acenderá se o sinal A* **NÃO** *estiver presente.*"

E a expressão correspondente será:

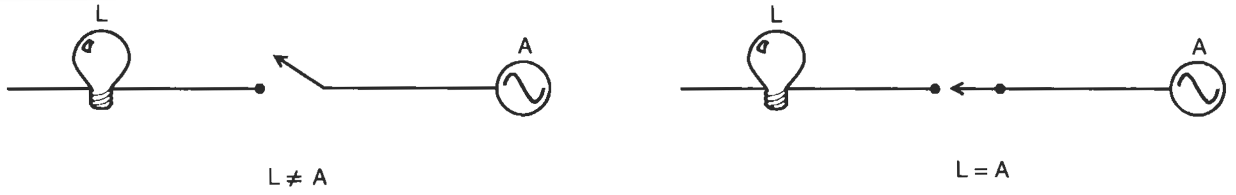
L = NOT A

Além de utilizar variáveis que possuem apenas dois valores, 0 e 1, a álgebra booleana define a existência de três operações fundamentais (as demais operações booleanas são combinações das três fundamentais):

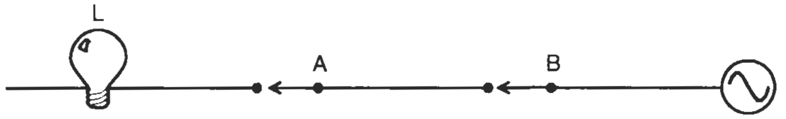
AND (representada pelo ponto, "·", da multiplicação aritmética)

OR (representada pelo sinal "+" da adição aritmética)

NOT (representada por uma barra em cima da variável, "$\overline{A}$", — terminologia ANSI — ou por apóstrofo à direita da variável "A'").



(a) Exemplo de afirmação expressa por uma equação lógica



(b) Exemplo de afirmação com dupla idéia

Figura 4.17 Exemplo de circuitos que implementam idéias.

Tendo em vista que livros, revistas e manuais, principalmente na área de computação, usam termos um pouco diferentes para identificar as operações lógicas AND, OR e NOT, parece interessante esclarecer o leitor mencionando estes termos.

Já vimos que (Fig. 4.6) a operação OR é representada pelo símbolo de + e que sua tabela verdade consiste nas linhas a seguir:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

Quem observa essas expressões, à primeira vista pode pensar que se trata de soma aritmética e, não fosse pela última operação, que os resultados são iguais mesmo aos de uma soma aritmética. É por isso que alguns autores chamam a operação lógica OR de soma lógica e outros criaram diferentes símbolos para não confundirmos com soma. Símbolos como ∪ e V (usados também em teoria dos conjuntos).

Assim, usar-se-ia:

$A \cup B \cup C$ em vez de $A + B + C$

Mas o pessoal de computação (e os livros e manuais estão aí para confirmar isso) continua preferindo usar o símbolo +, talvez por uma deferência a mais com George Boole, já que foi ele quem propôs originalmente o símbolo.

O mesmo acontece com a operação AND e seu símbolo. Como o símbolo da referida operação (ponto "·") também representa uma operação de multiplicação aritmética e os resultados da tabela verdade (ver Fig. 4.4) são idênticos aos da multiplicação, alguns chamam a operação lógica AND de multiplicação lógica.

Regras Básicas da Álgebra Booleana

A Tabela 4.1 apresenta todas as regras básicas da álgebra booleana [MANO 82] e [BART 91].

Tabela 4.1 Regras Básicas da Álgebra Booleana

1. $X + 0 = X$
2. $X + 1 = 1$
3. $X + X = X$
4. $X + \overline{X} = 1$
5. $X \cdot 0 = 0$
6. $X \cdot 1 = X$
7. $X \cdot \overline{X} = X$
8. $X \cdot X = 0$

—

—

9. $X = X$
10. $X + Y = Y + X$
11. $X \oplus X = 0$

12. $X \cdot Y = Y \cdot X$
13. $X + (Y + Z) = (X + Y) + Z$
14. $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
15. $X \cdot (Y + Z) = X \cdot Y + X \cdot Z$
16. $X + Y \cdot Z = X$
17. $X (X + Y) + X$
18. $(X + Y) \cdot (X + Z) = X + Y \cdot Z$
19. $X + \overline{X} \cdot Y = X + Y$
20. $X \cdot Y + Y \cdot \overline{Z} + \overline{Y} \cdot Z = X \cdot Y + Z$
21. $\overline{(X + Y)} = \overline{X} \cdot \overline{Y}$
22. $\overline{(X \cdot Y)} = \overline{X} + \overline{Y}$

As expressões simples, de 1 a 9, que usam apenas uma variável, podem ser facilmente provadas através do emprego das tabelas verdade já mostradas anteriormente. Aliás, todas as demais regras também podem ser comprovadas com o emprego de tabelas verdade, embora algumas tenham um pouco mais de complexidade do que as nove primeiras.

Por exemplo:

- na regra 1: se $X = 0$, então: $0 + 0 = 0 = X$; e
  se $X = 1$, então: $1 + 0 = 1 = X$
- na regra 6: se $X = 0$, então: $0 \cdot 1 = 0 = X$; e
  se $X = 1$, então: $1 \cdot 1 = 1 = X$
- na regra 8: se $X = 0$, então: $0 \cdot \overline{0} = 0 \cdot 1 = 0$; e
  se $X = 1$, então: $1 \cdot \overline{1} = 1 \cdot 0 = 0$

As regras 10 e 12 são conhecidas como *lei da comutatividade* (aprendemos esta lei na escola para o caso da adição e da multiplicação), significando que a ordem das parcelas não afeta o resultado da respectiva operação.

A regra 11 não consta das tabelas de regras básicas, mas resolvemos inseri-la devido à natural importância do circuito lógico XOR como zerador de valores de registradores, testador de igualdades entre palavras e outras aplicações. Sua confirmação é simples, com o emprego da tabela verdade da Fig. 4.13; o resultado de um XOR é sempre *igual a 1* para valores diferentes, e *igual a zero* para valores iguais.

As regras 13 e 14 constituem a *lei da associatividade,* seja para a soma lógica (OR), seja para a multiplicação lógica (AND). Isto é, o resultado não se altera se a operação desejada, de soma ou multiplicação — OR ou AND — for efetuada entre o segundo e o terceiro operandos e depois com o primeiro operando, ou se for efetuada entre o primeiro e o segundo operandos, para em seguida operar com o terceiro.

A regra 15, que utiliza a *lei da distributividade,* estabelece que o produto (AND) de um operando por um polinômio (parcelas internas usando operação OR) é igual à soma (OR) do produto do referido operando com cada elemento do polinômio.

A regra 16,

$$X + Y \cdot Z + X$$

é válida porque, usando a regra 15 temos:

$$X + Y \cdot Z = X \cdot (1 + Z)$$

E, de acordo com a regra 2, temos que: $1 + Z = 1$

Então,

$X + X \cdot Z = X \cdot 1$

E, pela regra 6, sabemos que: $X \cdot 1 = X$

No caso da regra 17, $X \cdot (X + Y)$, esta pode ser provada como se segue:

$X \cdot (X + Y) = X \cdot X + X \cdot Y$ (regra 15) $= X$ (regra 7) $+ X \cdot Y =$

$= X \cdot (1 + Y)$ (regra 15) $= X \cdot 1$ (regra 2) $= X$ (regra 6)

A regra 18, que não se aplica na álgebra comum, pode ser provada como se segue:

$(X + Y) \cdot (X + Z) = X \cdot X + X \cdot Z + X \cdot Y + Y \cdot Z$ (regra 15) $=$

$= X$ (regra 7) $+ X \cdot Z + X \cdot Y + Y \cdot Z = X + X \cdot Y + X \cdot Z + Y \cdot Z$

$= X \cdot (1 + Y) + Z \cdot (X + Y) = X \cdot 1$ (regra 2) $+ Z \cdot (X + Y) =$

$= X$ (regra 6) $+ Z \cdot (X + Y) = X + Z \cdot (X + Y) = X + X \cdot Z + Y \cdot Z$

$= X \cdot (1 + Z) + Y \cdot Z = X \cdot 1 + Y \cdot Z = X + Y \cdot Z$

A regra 19 pode ser provada, como as regras anteriores:

$$X + \overline{X} \cdot Y = \underbrace{(X + X \cdot Y)}_{\text{regra 16}} + \overline{X} \cdot Y = X + X \cdot Y + \overline{X} \cdot Y$$

$X + Y \cdot (X + \overline{X}) = X + Y \cdot 1$ (regra 4) $= X + Y$ (regra 6)

A regra 20 pode também ser provada de modo idêntico:

$X \cdot Y + Y \cdot Z + \overline{Y} \cdot Z = X \cdot Y + Z \cdot (Y + \overline{Y})$ regra 15 $=$

$= X \cdot Y + Z \cdot 1$ (regra 4) $= X \cdot Y + Z$ (regra 6) $= X \cdot Y + Z$

As regras 21 e 22 constituem, em conjunto, o Teorema de Morgan, e são repetidas em seguida:

$(\overline{X + Y}) = \overline{X} \cdot \overline{Y}$

$(\overline{X \cdot Y}) = \overline{X} + \overline{Y}$

Com este teorema pode-se obter o complemento de qualquer expressão.

"Esquecendo" sua demonstração nesse ponto, a conclusão mais importante é que, pelo teorema, pode-se substituir a expressão not $X \cdot$ not Y por uma simples porta NOR e a expressão not X + not Y por uma simples porta NAND, com sensível redução de custos. A expressão not $X \cdot$ not Y é, na realidade, idêntica ao seguinte circuito lógico: NOT X AND NOT Y, o qual utiliza três circuitos diferentes, duas portas NOT e uma porta AND. O Teorema de Morgan também pode reduzir esses três circuitos para apenas um, com o uso da porta NOR. O mesmo pode ser exposto para a outra expressão, que substitui 2 NOT e 1 OR por apenas 1 NAND.

Uma das mais vantajosas conseqüências do emprego da álgebra booleana está justamente na utilização de suas regras para simplificar as expressões lógicas que definem a função de um dado dispositivo digital. Essas simplificações não só facilitam o entendimento do funcionamento do dispositivo, como também contribuem para a redução dos custos de fabricação dos circuitos digitais, devido à redução de componentes eletrônicos utilizados.

Vejamos, em seguida, alguns exemplos do emprego da álgebra booleana na simplificação de expressões lógicas.

Exemplo 4.25

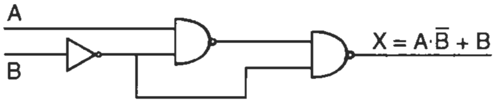
Simplificar a expressão: $X = [\overline{(\overline{A} + B) \cdot \overline{B}}]$

Solução

A Fig. 4.18 mostra a expressão a ser simplificada (4.18(a)), o diagrama lógico equivalente à referida expressão (4.18(b)) e a expressão simplificada resultante (4.18(c)), com o seu diagrama lógico correspondente (4.18(d)). A simples observação da figura permite concluir o quanto se obteve de economia de circuitos lógicos, passando-se dos 3 iniciais (2 NAND e 1 NOT) para apenas 1 (1 OR).

$X = \overline{[(\overline{A} + B)\cdot\overline{B}]}$

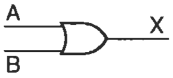
(a) Exemplo a ser simplificado



(b) Diagrama lógico correspondente à expressão em (a)

$X = A + B$

(c) Exemplo simplificado



(d) Diagrama lógico correspondente à expressão em (c)

Figura 4.18 Exemplo de simplificação de circuito lógico.

$X = \overline{[(\overline{A} + B)\cdot \overline{B}]}$

Usando o teorema de Morgan, teremos:

$X = \overline{(\overline{A} + B)} + \overline{\overline{B}}$ (usamos $\overline{X \cdot Y} = \overline{B} + \overline{Y}$)

Usando novamente o teorema, $\overline{(X + Y)} = \overline{X} \cdot \overline{Y}$ para a 1.ª parcela do OR:

$X = \overline{\overline{A}} \cdot \overline{B} + \overline{\overline{B}}$

Aplicando a regra 9 em $\overline{\overline{A}}$ e $\overline{\overline{B}}$, teremos:

$X = A \cdot \overline{B} + B$

Continuando a simplificação pelas regras da Tabela 4.1, teremos:

$X = A \cdot \overline{B} + B = B + A \cdot \overline{B}$ (regra 10)

$X = A + B$ (regra 19)

Se usássemos a prova da tabela verdade (o leitor interessado está convidado a fazê-lo, de modo semelhante ao do Exemplo 4.24), concluiríamos que ambas as expressões produzem os mesmos resultados.

Exemplo 4.26

Simplificar a expressão: $X = \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot C$

Solução

Utilizando as regras da Tabela 4.1, teremos:

$X = \overline{A} \cdot B. (\overline{C} + C) + A \cdot C \cdot (\overline{B} + B)$ regra 15

$X = \overline{A} \cdot B \cdot 1 + A \cdot C \cdot 1$ regra 4

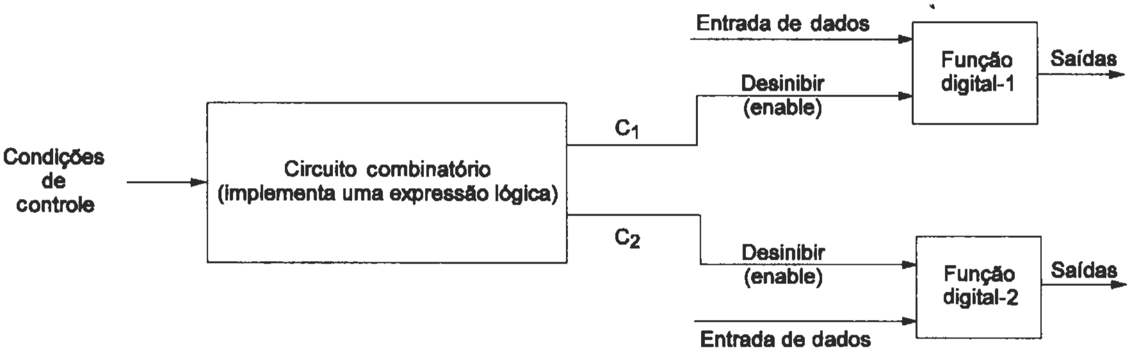
$X = \overline{A} \cdot B + A \cdot C$ regra 6

4.4.3 Circuitos Combinatórios

Após analisarmos cada porta individualmente (sua definição e os resultados obtidos com o emprego da tabela verdade de cada uma) e termos conhecido as regras que servem de alicerce à álgebra booleana e seu emprego para a simplificação de expressões lógicas, podemos verificar como é possível interligar as diversas portas, de modo a construir *redes lógicas* (*logic networks*), também chamadas de *circuitos combinatórios* (termo mais usado em teoria de chaveamento (*switching theory*), para projeto, análise e construção de circuitos de chaveamento (*switching circuits*)).

Um *circuito combinatório* é definido como um conjunto de portas cuja saída em qualquer instante de tempo é função somente das entradas [STAL 87]. Em contrapartida, um *circuito seqüencial,* além de possuir portas, contém elementos de armazenamento, denominados flip-flops (ver item 4.4.4.2).

A Fig. 4.19 mostra um diagrama exemplificando o emprego de circuitos combinatórios (rede lógica) [MANO 82]. No item 6.6.4.1 é mostrado o uso desses circuitos para fazer funcionar a unidade de controle das UCP (em oposição à metodologia de emprego de sistemas microprogramados).



Função digital-1: Realiza a soma de dois números que aparecem na sua entrada.

Função digital-2: Realiza a soma de um número e uma constante que aparecem na sua entrada.

Figura 4.19 Exemplo de uso de circuito combinatório.

Em resumo, o exemplo da Fig. 4.19 consiste no seguinte:

a) Os módulos da função digital 1 e 2 representam circuitos que realizam algum tipo de operação, como, por exemplo, soma de dois valores que cheguem à entrada (módulo 1) ou soma de um valor e uma constante (módulo 2).

b) Cada módulo é acionado para executar a ação que lhe corresponde quando um sinal de controle (C_1, para o módulo 1 e C_2, para o módulo 2) assume o valor 1 (ativado), nada acontecendo se o valor for igual a zero.

c) O circuito combinatório produzirá a saída adequada, de acordo com os sinais de tempo (condições de controle) que aparecem nas suas entradas (ver Fig. 6.43).

d) Supondo, por exemplo, que as condições de ativação da saída 1 do circuito combinatório sejam três seqüências de tempo, T_1, T_2 e T_3 , mais as variáveis binárias x e y, geradas internamente, os sinais de tempo ocorrem em seqüência, de modo que, em um instante de relógio, $T_1 = 1$ e $T_2 = T_3 = 0$; no período seguinte (outro pulso de relógio), $T_1 = 0$, $T_2 = 1$ e $T_3 = 0$; em seguida, $T_1 = T_2 = 0$ e $T_3 = 1$, e depois tudo se repete. As variáveis internas x e y são resultantes de outras operações previamente realizadas, co-mo, por exemplo, o resultado de uma operação aritmética ser igual a zero ($X = 1$) ou diferente de zero ($X = 0$).

e) As condições que definem o valor das variáveis de controle C_1 e C_2 podem ser especificadas por expressões booleanas (lógicas). Vamos, por exemplo, considerar as seguintes expressões (tarefa das mais complexas para os projetistas de Unidades de Controle do tipo programadas por hardware ou *hardwired*):

$C_1 = x \cdot T_1 + T_2 + y \cdot T_3$

$C_2 = y \cdot T_3$

f) A interpretação do processo consistiria no seguinte:

1) se $x = 1$ **e** T_1 **ou** $T_2 = 1$ **ou** $Y = 1$ **e** $T_3 = 1$,
   então: $C_1 = 1$,
   senão: $C_1 = 0$;
2) quando $C_1 = 1$, a função digital 1 é realizada com os dados de entrada;
3) se $Y = 0$ **e** $T_3 = 1$,
   então: $C_2 = 1$
   senão: $C_2 = 0$;
4) quando $C_2 = 1$, a função digital 2 é realizada com os dados de entrada.

g) A Fig. 4.20 mostra o diagrama lógico do circuito combinatório aqui exemplificado.

Entre outros exemplos de circuitos combinatórios, podemos citar o somador parcial, descrito no item 6.6.1, e o somador completo, também descrito no mesmo item (ver Fig. 6.33).

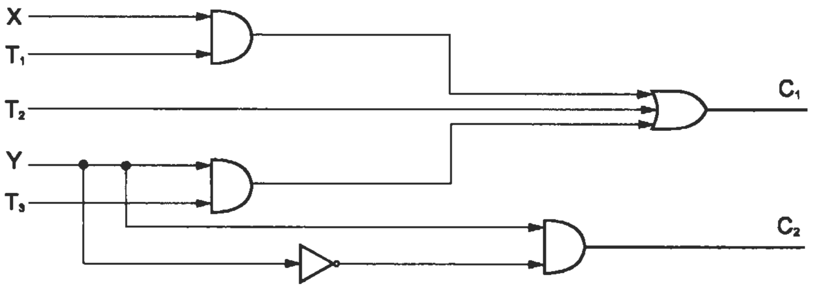


Figura 4.20 Diagrama lógico do circuito combinatório da Fig. 4.19.

4.4.3.1 Exemplo Prático — Projeto de um Multiplicador de 2 Bits

Para compreender melhor como a álgebra booleana é utilizada na implementação de um problema prático, vamos descrever um exemplo, apresentado em [CLEM 86], consistindo no projeto de um circuito combinatório que recebe dois valores numéricos de 2 bits cada e produz, na saída, um valor de 4 bits igual ao produto dos valores de entrada, isto é, um multiplicador de 2 bits.

A Fig. 4.21 apresenta o diagrama em bloco do multiplicador, contendo as quatro entradas, para o número A, com 2 bits, e para o número B, com 2 bits, e a saída com quatro linhas, uma para cada bit do valor do resultado, R.

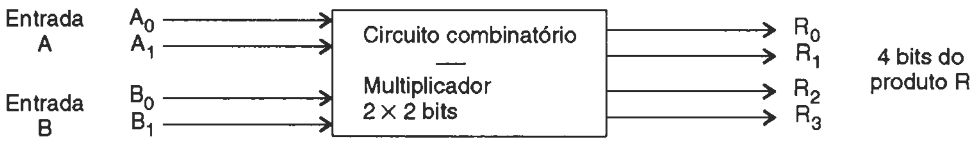


Figura 4.21 Diagrama em bloco de um multiplicador de 2 bits.

Como temos 4 bits na entrada (2 para cada número), podemos construir a tabela verdade da multiplicação com 16 combinações possíveis ($2^4 = 16$), conforme mostrado na Tabela 4.2.

Observando a Tabela 4.2, verificamos que ela possui 16 linhas de informações porque há 4 bits de entrada, A_0, A_1, B_0 e B_1 e, portanto, tem 16 combinações entre 0000 e 1111. Cada uma dessas combinações representa um produto mostrado na coluna da esquerda (A*B = R), como, por exemplo, 1 × 3 = 3, que é o mesmo que $A_0 = 0$, $A_1 = 1$, $B_0 = 1$ e $B_1 = 1$, e tendo como resultado: $R_0 = 0$, $R_1 = 0$, $R_2 = 1$ e $R_3 = 1$.

Tabela 4.2 Tabela Verdade para a Multiplicação de 2 Valores Binários

A * B + R	Entradas				Saídas			
	A		B		R			
	A_0	A_1	B_0	B_1	R_0	R_1	R_2	R_3
0 * 0 = 0	0	0	0	0	0	0	0	0
0 * 1 = 0	0	0	0	1	0	0	0	0
0 * 2 = 0	0	0	1	0	0	0	0	0
0 * 3 = 0	0	0	1	1	0	0	0	0
1 * 0 = 0	0	1	0	0	0	0	0	0
1 * 1 = 1	0	1	0	1	0	0	0	1
1 * 2 = 2	0	1	1	0	0	0	1	0
1 * 3 = 3	0	1	1	1	0	0	1	1
2 * 0 = 0	1	0	0	0	0	0	0	0
2 * 1 = 2	1	0	0	1	0	0	1	0
2 * 2 = 4	1	0	1	0	0	1	0	0
2 * 3 = 6	1	0	1	1	0	1	1	0
3 * 0 = 0	1	1	0	0	0	0	0	0

Para criar o circuito combinatório que implemente a tabela verdade (tabela 4.2), devem ser construídas quatro expressões lógicas, uma para cada saída. Na realidade, um circuito com X saídas terá sempre X expressões lógicas.

$$R_0 = A_0 \cdot \overline{A}_1 \cdot B_0 \cdot \overline{B}_1 + A_0 \cdot \overline{A}_1 \cdot B_0 \cdot \overline{B}_1 + A_0 \cdot A_1 \cdot B_0 \cdot \overline{B}_1 + A_0 \cdot A_1 \cdot B_0 \cdot B_1$$

que pode ser simplificada para:

$$R_0 = A_0 \cdot \overline{A}_1 \cdot B_0 \cdot (B_1 + \overline{B}_1) + A_0 \cdot A_1 \cdot B_0 \cdot (B_1 + \overline{B}_1) = A_0 \cdot \overline{A}_1 \cdot B_0 + A_0 \cdot A_1 \cdot B_0 =$$

$$= A_0 \cdot B_0\,(A_0 + \overline{A}_1) =$$

$$R_0 = A_0 \cdot B_0$$

$$R_1 = A_0 \cdot \overline{A}_1 \cdot \overline{B}_0 \cdot B_1 + A_0 \cdot \overline{A}_1 \cdot B_0 \cdot B_1 + \overline{A}_0 \cdot A_1 \cdot B_0 \cdot \overline{B}_1 + \overline{A}_0 \cdot A_1 \cdot B_0 \cdot B_1 + A_0 \cdot A_1 \cdot B_0 \cdot \overline{B}_1 +$$

$$A_0 \cdot A_1 \cdot \overline{B}_0 \cdot B_1 =$$

$$= A_0 \cdot \overline{A}_1 \cdot B_1 \cdot (\overline{B}_0 + B_0) + \overline{B}_0 \cdot B_1 \cdot B_0\,(\overline{B}_1 + B_0) + A_0 \cdot A_1 \cdot B_0 \cdot \overline{B}_1 + A_0 \cdot A_1 \cdot \overline{B}_0 \cdot B_1 =$$

$$= A_0 \cdot \overline{A}_1 \cdot B_1 + \overline{A}_0 \cdot A_1 \cdot B_0 + A_0 \cdot A_1 \cdot B_0 \cdot \overline{B}_1 + A_0 \cdot A_1 \cdot \overline{B}_0 \cdot B_1 =$$

$$= A_0 \cdot B_1 \cdot (\overline{A}_1 + A_1 \cdot B_0) + A_1 \cdot B_0 \cdot (\overline{A}_0 + A_0 \cdot \overline{B}_1) =$$

$$= A_0 \cdot B_1 \cdot (\overline{A}_1 + \overline{B}_0) + A_1 \cdot B_0 \cdot (\overline{A}_0 + B_1) =$$

$$R_1 = A_0 \cdot \overline{A}_1 \cdot B_1 + A_0 \cdot \overline{B}_0 \cdot B_1 + \overline{A}_0 \cdot A_1 \cdot B_0 + A_1 \cdot B_0 \cdot \overline{B}_1$$

$$R_2 = \overline{A}_0 \cdot A_1 \cdot \overline{B}_0 \cdot B_1 + A_1 \cdot \overline{A}_0 \cdot B_1 \cdot B_0 + A_0 \cdot A_1 \cdot \overline{B}_0 \cdot B_1 =$$

$$= \overline{A}_0 \cdot A_1 \cdot B_1 \cdot (\overline{B}_0 + B_0) + A_0 \cdot A_1 \cdot \overline{B}_0 \cdot B_1 =$$

$$= \overline{A}_0 \cdot A_1 \cdot B_1 + A_0 \cdot A_1 \cdot \overline{B}_0 \cdot B_1 =$$

$= A_1 \cdot B_1 \cdot (\overline{A}_0 + A_0 \cdot \overline{B}_0) =$

$= A_1 \cdot B_1 \cdot (\overline{A}_0 + \overline{B}_0) =$

$R_2 = \overline{A}_0 \cdot A_1 \cdot B_1 + A_1 \cdot \overline{B}_0 \cdot B_1$

$R_3 = A_0 \cdot A_1 \cdot B_0 \cdot B_1$

Em resumo, as 4 expressões lógicas correspondentes às 4 saídas são:

$R_0 = A_0 \cdot B_0$

$R_1 = A_0 \cdot \overline{A}_1 \cdot B_1 + A_0 \cdot \overline{B}_0 \cdot B_1 + \overline{A}_0 \cdot A_1 \cdot B_0 + A_1 \cdot B_0 \cdot \overline{B}_1$

$R_2 = \overline{A}_0 \cdot A_1 \cdot B_1 + A_1 \cdot \overline{B}_0 \cdot B_1$

$R_3 = A_0 \cdot A_1 \cdot B_0 \cdot B_1$

A Fig. 4.22 mostra uma entre várias possíveis implementações das expressões lógicas R_0 a R_3.

É possível simplificar certas expressões lógicas, transformando-as somente em circuitos NAND. Isto é desejável em certas circunstâncias porque a porta NAND é mais rápida e mais barata que o correspondente circuito AND. O mesmo, aliás, acontece com os circuitos lógicos NOR e OR, onde o primeiro é também mais barato e rápido que este último.

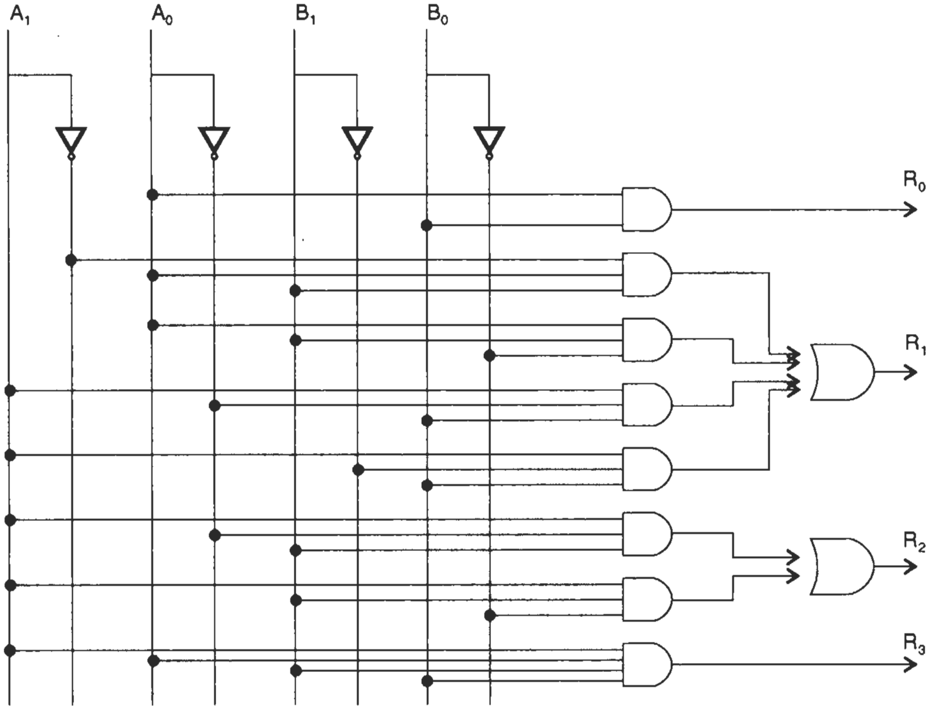


Figura 4.22 Uma implementação das expressões lógicas R_0 a R_3 do exemplo da Fig. 4.21.

4.4.3.2 Portas Wired-Or e Wired-And

Um outro exemplo de utilização de circuitos combinatórios se refere a uma certa maneira de construir várias portas OR ou AND por meio de uma única conexão (ver item 5.6.1.5 para um exemplo de aplicação dessa tecnologia).

A Fig. 4.23 mostra uma combinação de porta NAND para AND, na qual a porta AND é formada pela conexão das saídas das portas NAND. Neste circuito (wired-AND) não há necessidade de nenhum outro elemento a não ser as portas NAND, o que é comprovado pelas linhas tracejadas da figura [BART 91].

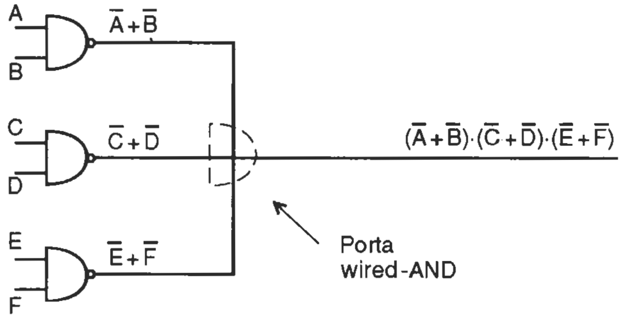


Figura 4.23 Exemplo de implementação de circuito wired-AND

Nem todas as portas NAND podem ter suas saídas conectadas dessa maneira para formar uma porta AND. Antes de sua obtenção, é preciso verificar se o tipo desejado é o adequado.

A Fig. 4.24 mostra um circuito lógico composto de portas NOR que formam um OR (wired-OR) de modo semelhante ao exemplo anterior para o circuito wired-AND.

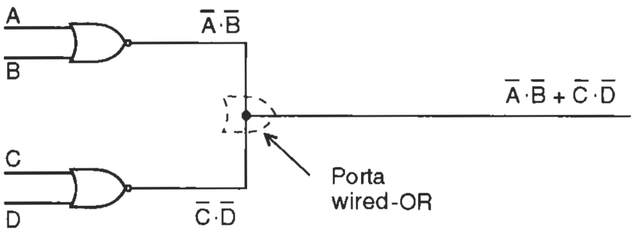


Figura 4.24 Exemplo de implementação de circuito wired-OR.

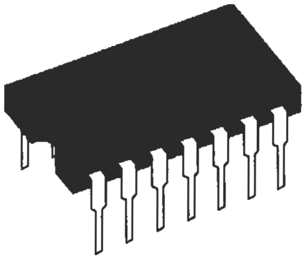
4.4.4 Circuitos Integrados

Após a primeira parte do processo, o projeto de um circuito combinatório, completa-se esta tarefa com a implementação física do circuito através da fabricação dos elementos e integrando-os em um só dispositivo (encapsulando-o), com o propósito de redução de espaço e custo, isto é, construindo um *circuito integrado.*

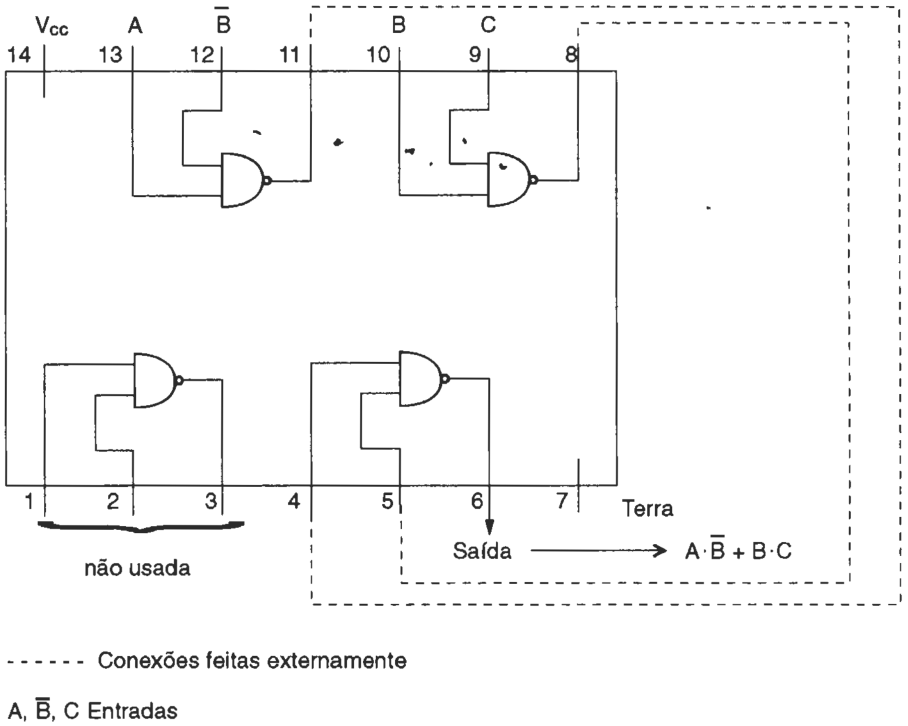
Um circuito integrado — CI (IC — Integrated Circuit) é um pequeno dispositivo, denominado *pastilha* (*chip*), que contém em seu interior centenas e atualmente milhares de componentes eletrônicos: transistores, diodos, resistores, capacitores e suas interligações. Estes componentes são os formadores das portas lógicas que, interligadas, formam um determinado circuito combinatório.

A pastilha é encapsulada em um pacote de cerâmica ou plástico e as conexões com o exterior são soldadas aos pinos externos para completar o dispositivo. A Fig. 4.25(a) mostra uma pequena pastilha de circuito integrado (CI) e a Fig. 4.25(b), o diagrama lógico de seu interior.

A pastilha é mostrada com 14 pinos, sendo alguns tipos inseridos e outros soldados a um componente maior, normalmente uma placa de circuito impresso.



(a) Figura da pastilha de um circuito integrado com 14 pinos


V_{CC}
A
$\overline{B}$
B
C
14
13
12
11
10
9
8
1
2
3
4
5
6
7
Terra
não usada
Saída
$A \cdot \overline{B} + B \cdot C$
- - - - - Conexões feitas externamente
A, $\overline{B}$, C Entradas

(b) Diagrama lógico do circuito integrado

Figura 4.25 Exemplo de um pequeno circuito integrado.

É possível se construir pastilhas com pequena quantidade de portas lógicas (pequena integração) até pastilhas de uma UCP (como a dos processadores Pentium, fabricados pela Intel, ou K7, fabricado pela AMD) contendo milhões de transistores (integração em muito larga escala — VLSI). Pastilhas de CI (IC) podem ser fabricadas com diferentes quantidades de pinos. Os fabricantes de CI costumam identificar cada um por um número, além da função à qual se destinam, e os fabricantes de sistemas de computação adquirem, então, estes CI de acordo com suas necessidades, para serem inseridos nas placas de circuito impresso. Este processo (fabricantes especializados na construção de circuitos integrados padronizados para executar determinadas funções) tira um pouco da flexibilidade do fabricante de sistemas de computação, visto que ele deve projetar seu sistema usando as pastilhas que já existem no mercado e não fazendo, talvez, tudo o que se desejaria para o referido sistema. No entanto, apesar deste inconveniente, isso ainda é mais vantajoso do que o próprio fabricante do computador ter que também verticalizar, projetando e construindo todas as pastilhas necessárias (seria um processo semelhante ao das fábricas de automóveis, que na realidade são denominadas montadoras, que adquirem no mercado a quase totalidade das peças necessárias à montagem do veículo, em vez de a própria montadora fabricar todas as peças).

É comum classificar pastilhas (*chips*) pela quantidade de integração que elas suportam, isto é, a quantidade de portas lógicas (e, em última análise, a quantidade de transistores encapsulados):

- *SSI* (*Small Scale Integration*) — Integração em pequena escala constituída de 1 a 10 portas. Poucos pinos. O exemplo da Fig. 4.25 é uma amostra de pastilha SSI de número 7400, enquanto a Fig. 4.26 apresenta outros exemplos de pastilhas SSI.

Todas as pastilhas apresentadas na figura possuem 14 pinos, 7 de cada lado, caracterizando-se como um pacote DIL (*Dual-Inline*), também chamado DIP (*Dual-Inline Package*).

- *MSI* (*Medium Scale Integration*) — Integração em média escala constituída de até 100 portas. Tem também poucos pinos. Circuitos desse tipo costumam ser usados para a construção de certos pequenos dispositivos, como multiplicadores e contadores.
- *LSI* (*Large Scale Integration*) — Integração em larga escala constituída de até 100.000 portas. Normalmente são processadores completos.
- *VLSI* (*Very Large Scale Integration*) — Integração em muito larga escala. É costume classificar pastilhas com até 100.000 portas como LSI e, acima desta quantidade, como VLSI. O processador Intel 80486 possui cerca de 1,2 milhão de transistores em uma única pastilha, com 168 pinos, enquanto o processador Pentium possui cerca de 3,1 milhões de transistores (no caso, trata-se dos modelos iniciais de Pentium, pois os atuais modelos possuem muito mais transistores na pastilha) em uma pastilha de CI com 273 pinos.

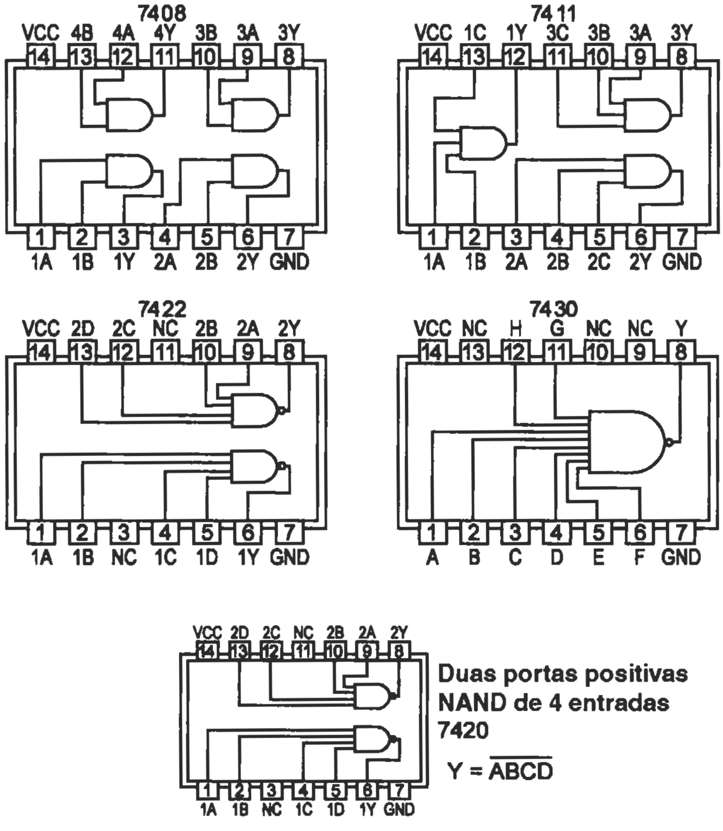

7408
VCC 4B 4A 4Y 3B 3A 3Y
14 13 12 11 10 9 8
1 2 3 4 5 6 7
1A 1B 1Y 2A 2B 2Y GND
7411
VCC 1C 1Y 3C 3B 3A 3Y
14 13 12 11 10 9 8
1 2 3 4 5 6 7
1A 1B 2A 2B 2C 2Y GND
7422
VCC 2D 2C NC 2B 2A 2Y
14 13 12 11 10 9 8
1 2 3 4 5 6 7
1A 1B NC 1C 1D 1Y GND
7430
VCC NC H G NC NC Y
14 13 12 11 10 9 8
1 2 3 4 5 6 7
A B C D E F GND
VCC 2D 2C NC 2B 2A 2Y
14 13 12 11 10 9 8
1 2 3 4 5 6 7
1A 1B NC 1C 1D 1Y GND
Duas portas positivas
NAND de 4 entradas
7420
Y = $\overline{ABCD}$

Figura 4.26 Exemplos de CI do tipo SSI.

Outro tipo de classificação de circuitos integrados se refere à tecnologia do circuito eletrônico básico utilizado para montagem do circuito combinatório correspondente. A topologia do circuito básico, os compo-

BIBLIOTECA UNIBAN
Universidade Bandeirante de São Paulo

nentes eletrônicos utilizados e suas interligações caracterizam cada classificação diferente. A Tabela 4.3 mostra algumas dessas tecnologias atualmente em uso [BART 91]:

Tabela 4.3 Terminologias de Fabricação de Circuitos Integrados

Código	Descrição
74XXX	Família TTL original
74LXXX	Versão de baixo consumo do TTL padrão
74LSXXX	TTL Schottky de baixa potência. Largamente usado
74SXXX	Antigo Schottky (TTL)
74ALXXX	Schottky avançado, de baixa potência
74ASXXX	Schottky avançado
74CXXX	CMOS equivalente aos TTL
74HCXXX	CMOS de alta velocidade
74HCTXXX	CMOS de alta velocidade compatível com TTL
74ACXXX	CMOS avançado
74ACTXXX	CMOS avançado, compatível com TTL
ECL	Alta velocidade, alta potência, baixa densidade, somente MSI
GaAs	Gallium-Arsenide. Alta velocidade. Caro. Baixa densidade

Obs.: XXX indica 3 dígitos. Para cada código, o CI possui a mesma pinagem. Por exemplo, o CI 74166 tem a mesma pinagem do CI 74S166 e do CI 74ALS166.

Por fugir ao escopo deste livro, não entraremos em detalhes sobre o processo de fabricação de cada tipo. O leitor interessado poderá recorrer a [BART 81], [BART 91] e [MCCCL 86] para uma descrição mais pormenorizada do assunto. Ainda assim, é possível fazer algumas observações interessantes:

a) Há dois tipos de transistores usados em circuitos digitais: *bipolar* e *FET — Field Effect Transistor.* Embora, neste texto, não seja possível analisar cada um desses tipos, pode-se mencionar que a técnica bipolar usa transistores convencionais e que a técnica FET emprega um processo não-convencional.

b) Os circuitos integrados da Tabela 4.3, exceto os que usam tecnologia CMOS (74C, 74HC, 74HCT, 74AC e 74ACT), são do tipo "lógica bipolar", enquanto os CI do tipo CMOS usam tecnologia MOS — Metal Oxide Semiconductor e empregam transistores do tipo FET.

c) A lógica bipolar é muito usada quando se deseja alta velocidade dos dispositivos e o custo não é problema (são mais caros que os FET). Linhas com lógica bipolar consomem mais corrente e calor, razão por que não se costuma usar muitas portas em um encapsulamento deste tipo (são MSI).

d) Para se produzir circuitos integrados em larga e muito larga escala (LSI e VLSI), costuma-se empregar transistores FET, e a tecnologia básica empregada na fabricação do CI é a MOS (Metal Oxide Semiconductor). Transistores FET construídos com técnica MOS são chamados MOSFET.

e) Apesar de os transistores FET não serem tão rápidos quanto os de tecnologia bipolar, eles são menores (podem-se integrar mais elementos), de fabricação simples e de baixo custo, consomem menos corrente e calor (também auxiliam na integração de mais elementos). Por isso, são mais empregados quando se desejam muitos elementos, como grandes memórias, microprocessadores etc.

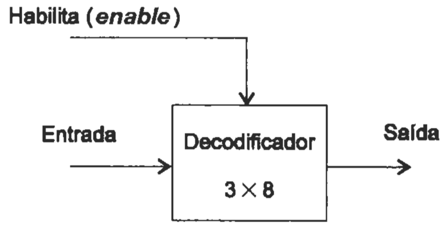
4.4.4.1 Decodificador

É um tipo de circuito combinatório de média integração (MSI), que possui x linhas de entrada e 2^x linhas de saída. Para cada configuração de bits que aparece na entrada haverá uma e somente uma linha de saída ativa, definida conforme o padrão de bits de entrada. Por exemplo, um decodificador pode ter 3 linhas de entrada ($x = 3$) e 8 linhas de saída ($2^3 = 8$), o que significa que na entrada podem aparecer valores entre 000 e 111, um valor de cada vez. Quando aparecer na entrada o valor 100, será ativada (a linha aparece com um pulso de voltagem correspondente ao bit 1) a 5.ª linha de saída e se aparecer na entrada o valor 010, será ativada a 3.ª linha de saída.

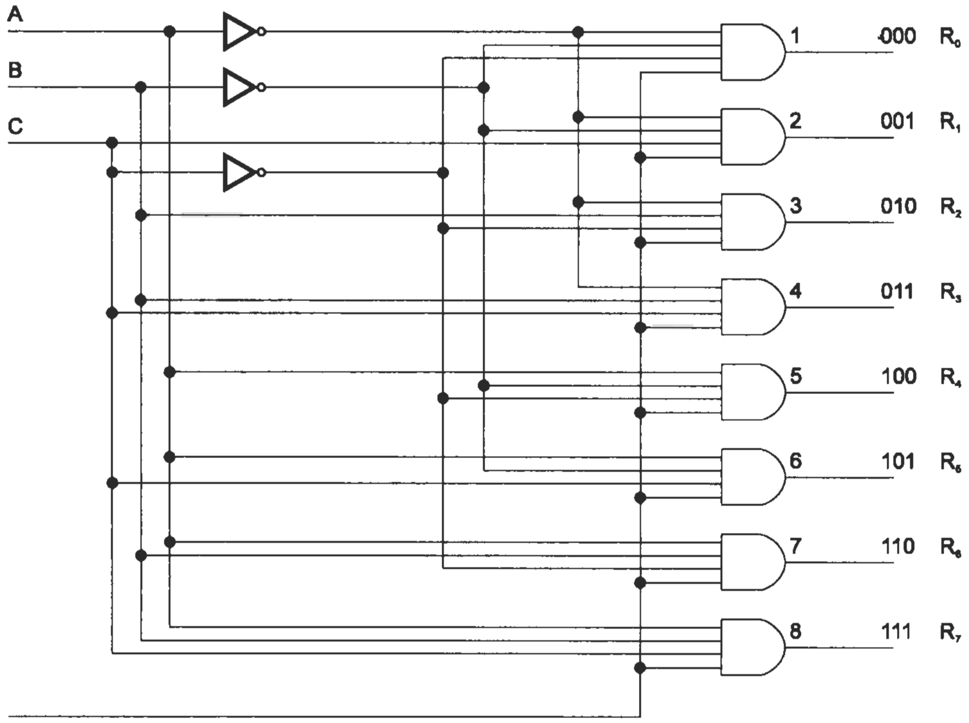
A Fig. 4.27 mostra um exemplo de decodificador 3 × 8, isto é, com 3 linhas de entrada e 8 linhas de saída, contendo a sua tabela verdade (Fig. 4.27(a)), o diagrama lógico do decodificador (Fig. 4.27(b)) e o seu diagrama em bloco (Fig. 4.27(c)). Normalmente, é mais comum representar o decodificador pelo seu diagrama em bloco do que pelo diagrama lógico, de modo a se reduzir a complexidade do desenho.

A	B	C	R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0

(a) Tabela verdade



(c) Diagrama em bloco



(b) Diagrama lógico

Figura 4.27 Decodificador 3 × 8, com linha habilita (*enable*).

Podem ser fabricados decodificadores com ou sem sinal de habilitação (*enable*) na entrada. Quando o sinal existe, ele serve para permitir ou não o aparecimento do valor de saída. Se a linha de habilitação for igual a 0, todas as saídas serão iguais a 0, porque a linha está conectada a cada porta AND e esta porta produzirá saída 0 se pelo menos uma entrada for igual a 0. Se a linha de habilitação for igual a 1, então o decodificador operará normalmente.

Se, como mencionamos anteriormente, aparece na entrada o valor 100, isto é, A = 1, B = 0 e C = 0, então teremos na entrada de cada porta AND e na saída os valores mostrados na Tabela 4.4.

A tabela mostra que a única saída ativada (bit 1) é a da 5.ª coluna (M), correspondente a 100, o valor de entrada.

Tabela 4.4 Valores de Entrada e de Saída de um Decodificador

Entradas	R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7
1.ª = A	0	0	0	0	1	1	1	1
2.ª = B	0	0	1	1	0	0	1	1
3.ª = C	0	1	0	1	0	1	0	1
Saídas	0	0	0	0	1	0	0	0

Uma boa aplicação desses dispositivos consiste na decodificação de instruções em uma UCP, servindo o código de operação como entrada do decodificador. Outra aplicação reside na decodificação de endereços para acesso à memória RAM.

A Fig. 4.28 mostra um diagrama em bloco de um decodificador usado para decodificar endereços. No exemplo, temos uma memória composta de 1024 (1K) bytes (cada célula ocupa 1 byte), organizados em 4 pastilhas de 256K cada. Cada endereço de memória é um número com 10 bits e, no caso, os 10 bits são divididos em duas partes. Os 8 bits menos significativos compreendem o endereço do byte desejado de uma das 4 pastilhas e os 2 bits mais significativos (mais à esquerda) indicam a qual das 4 pastilhas se está referindo. O decodificador 2 × 4 atua nesta última parte, isto é, para identificar, pela sua única saída válida, qual a pastilha que está sendo localizada.

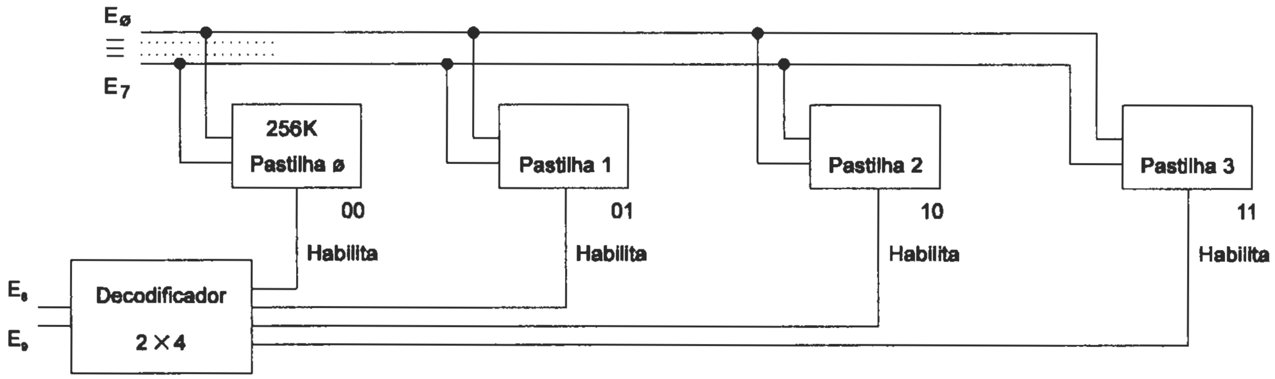


Figura 4.28 Decodificador 2 × 4, usado para decodificação de endereço.

4.4.4.2 Flip-Flops

O flip-flop não é construído como um circuito combinatório. Ele é, na realidade, um outro tipo de circuito lógico, denominado seqüencial. Um circuito seqüencial é constituído de um conjunto de flip-flops e portas lógicas interligadas. A interligação apenas de portas constitui, como já vimos, um circuito combinatório, mas quando se incluem flip-flops na conexão, então o chamamos de circuito seqüencial.

Se analisarmos de modo simplista os dois elementos que constituem um circuito seqüencial, os flip-flops e as portas lógicas, podemos imaginar os primeiros compondo uma memória e as últimas constituindo as operações que são realizadas em um computador.

Dessa maneira, o flip-flop é o elemento básico utilizado para se armazenar informações em um sistema digital. Ele é capaz de armazenar (operação de escrita) e permitir a leitura de um valor binário (bit 0 ou bit 1). São duas as características fundamentais de um circuito flip-flop:

a) É um elemento biestável, isto é, pode guardar (armazenar) um entre dois possíveis valores (correspondentes aos bits 0 e 1) permanentemente, enquanto estiver energizado. Em outras palavras, se um sinal de entrada acarreta sua passagem para um estado correspondente ao bit 1 (*set*), ele permanece neste estado até que outro sinal na entrada altere seu valor para 0.

b) Possui dois sinais, um deles sendo o complemento do outro (1 é complemento de 0 e vice-versa).

Flip-Flop do tipo S-R

A Fig. 4.29 mostra um diagrama em bloco de um circuito flip-flop, denominado flip-flop SR (há vários tipos de flip-flops produzidos para o mercado) ou latch SR.

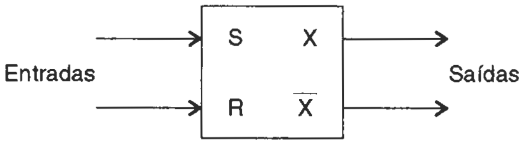


Figura 4.29 Diagrama em bloco de um flip-flop SR.

O estado do flip-flop (se é bit 1 ou bit 0) é obtido de sua saída X, ou seja, se a linha de saída marcada X está com sinal 1, diz-se que o flip-flop está no estado 1 e vice-versa. Além disso, as linhas de saída X e x são complemento uma da outra, de modo que, se X está com o valor 1, então x está com o valor 0 e vice-versa.

As linhas de entrada do flip-flop SR controlam o estado do flip-flop, como se segue:

a) se ambas as linhas de entrada estiverem com sinais equivalentes ao bit 0, então o flip-flop permanecerá no seu estado atual;

b) se a linha S (*set*) mudar para o estado 1 e a linha R mantiver o valor 0, então o flip-flop passará para o estado 1, isto é, diz-se que ele foi "setado";

c) quando a linha S estiver no estado 0 e a linha R (*reset*) estiver no estado 1, então o flip-flop passará para o estado 0; e

d) se ambas as linhas passarem para o estado 1, ocorrerá uma instabilidade, visto que as duas linhas não poderão simultaneamente ser "setadas" e "liberadas". A Fig. 4.30 mostra a tabela de estado do flip-flop RS.

	S	R	X	$\overline{X}$
Estado inicial de X = 0	0	0	0	1
Altera estado	1	0	1	0
Mantém estado anterior	0	0	1	0
Altera estado	0	1	0	1
Mantém estado anterior	0	0	0	1
Incorreto	1	1	-	-

X - estado do flip-flop após entrada R e S

$\overline{X}$ - complemento do valor de X

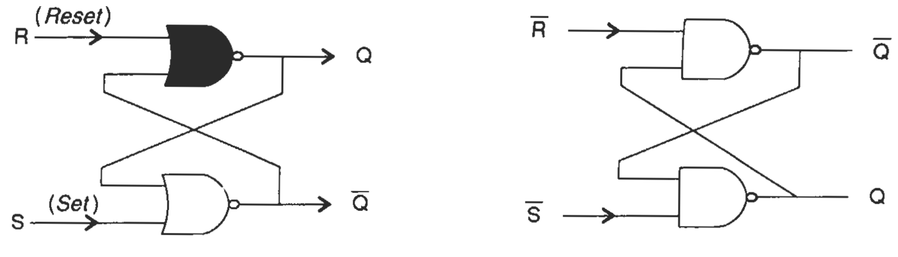
S - entrada *set*

R - entrada *reset*

Figura 4.30 Tabela de valores de um circuito flip-flop.

Circuitos flip-flops são construídos com portas lógicas, isto é, utilizando operadores lógicos interligados, como, por exemplo, com portas NOR, como mostra a Fig. 4.31(a), ou por circuitos lógicos NAND, como mostra a Fig. 4.31(b). A diferença de funcionamento no caso do flip-flop RS com portas NAND é que as entradas estarão complementadas, de modo que a tabela da Fig. 4.30 (que é válida para o caso de flip-flops RS com portas NOR) seria invertida. Por exemplo, se a entrada S for igual a 0 e a entrada R for igual a 1, então o flip-flop será "setado" no estado 1, e assim por diante.

Um registrador ou uma célula de memória pode ser construído com esse tipo de circuito, como veremos mais adiante. Cada um dos círculos funciona como uma célula de 1 bit. O valor apresentado na saída Q significa o valor do bit armazenado e as entradas R e S servem para que se possa escrever, respectivamente, o valor 1 ou 0. A Fig. 4.32 mostra um diagrama de tempo de um processo de escrita de um valor na célula de 1 bit, o flip-flop da Fig. 4.31(a), por exemplo. Em dado instante, o estado do flip-flop da figura é 0, ou seja,



(a) Com portas NOR

(b) Com portas NAND

Figura 4.31 Flip-flop SR.

$Q = 0$ e $\overline{Q} = 1$ e, nas entradas, tem-se $S = 0$ e $R = 0$, que é o instante A, na Fig. 4.32. Após um certo tempo, um valor 1 aparece na entrada S (*set*), instante B na Fig. 4.32 (o flip-flop é "setado"), que combina com o valor $Q = 0$ já existente na entrada da porta NOR inferior (Fig. 4.31(a)). Após um certo atraso, Δt (ponto C da Fig. 4.32), a saída da porta NOR inferior será igual a $\overline{Q} = 0$ (ver tabela verdade da porta NOR, Fig. 4.11, onde $S = 1$ NOR $Q = 0$ produz $\overline{Q} = 0$). Nesse mesmo instante C, a porta NOR superior terá como entrada os valores $R = 0$) (já estava) e $\overline{Q} = 0$ (acabou de acontecer no NOR inferior). Então, após um atraso Δt (a partir do instante C), que é igual a $2\ \Delta t$, a partir do instante B em que o valor 1 foi aplicado em S, a saída Q passa a ter valor 1 (instante D na Fig. 4.32). Este é o novo estado estável do flip-flop, tendo mudado de $Q = 0$ (estado anterior) para $Q = 1$ (estado atual) devido ao valor 1 aplicado em S. Houve, então, a escrita do valor 1 no flip-flop.

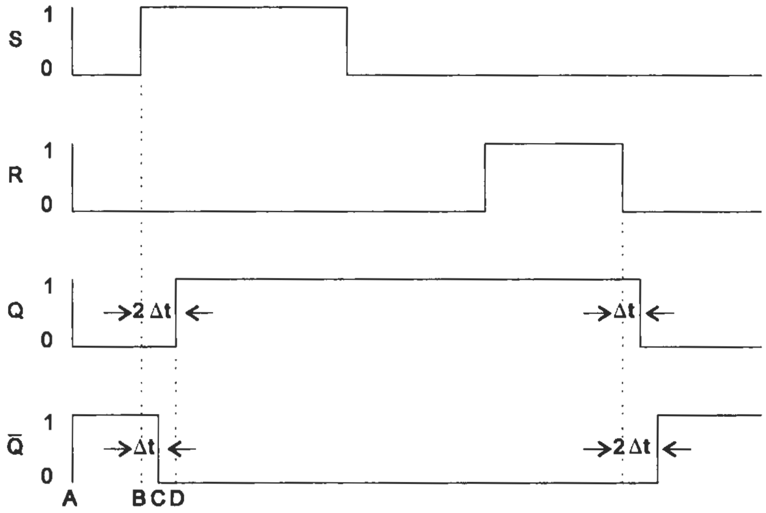
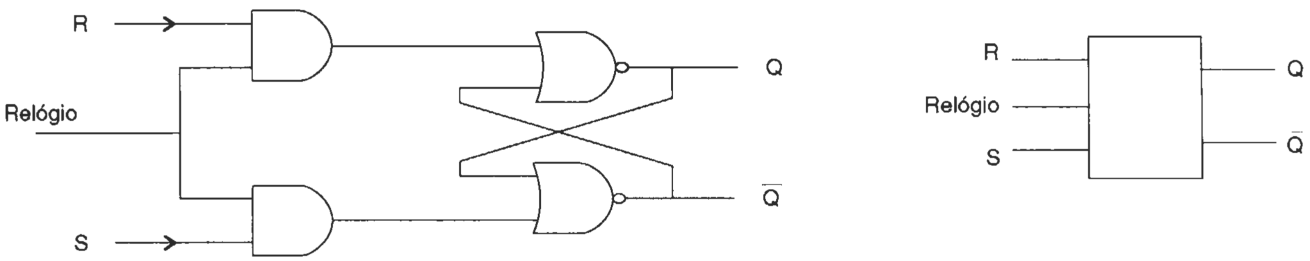


Figura 4.32 Diagrama de tempo de funcionamento de um flip-flop SR.

Flip-Flop S-R com Relógio

Conforme iremos observar no Cap. 6, um computador digital funciona basicamente através da realização de uma infindável quantidade de eventos, cuja iniciação depende dos regulares pulsos de relógio gerados na unidade de controle. Por essa razão, é conveniente acrescentar uma entrada de relógio ao flip-flop SR analisado anteriormente.

A Fig. 4.33 mostra um circuito SR com relógio, constituído de portas NOR e AND, sendo o diagrama lógico apresentado na Fig. 4.33(a) e o diagrama em bloco do mesmo circuito, na Fig. 4.33(b). Nesse circuito, as entradas R e S somente aparecem na entrada das portas NOR (para “setarem” ou “ressetarem” o flip-flop) quando é aplicado na entrada do flip-flop (portas AND) um pulso de relógio (valor igual a 1). Nesse momento, os valores de R e S, combinados, em cada AND, com o valor 1, produzem o mesmo resultado na saída dos circuitos AND.



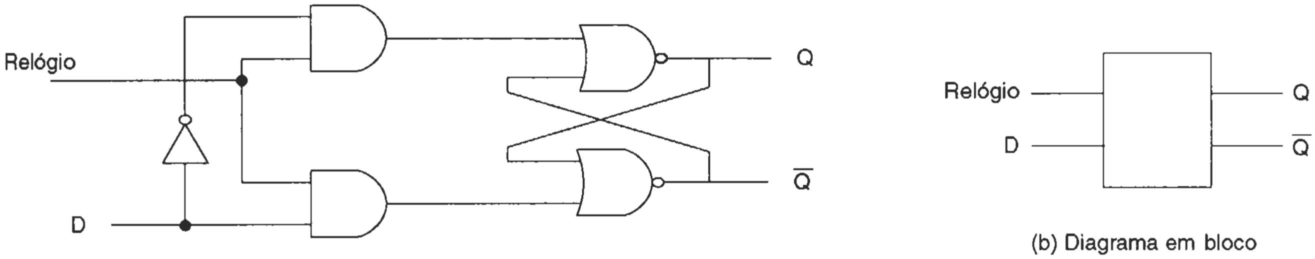
(b) Diagrama em bloco

(a) Diagrama lógico

Figura 4.33 Flip-flop com relógio.

Flip-Flop do tipo D

Quando descrevemos o circuito flip-flop RS foi observado que o circuito poderia assumir uma certa instabilidade no caso de ambas as entradas serem iguais a 1. Para evitar essa inconsistência, foi desenvolvido um outro tipo de circuito flip-flop, com apenas uma entrada (o que evita o problema). Este circuito é denominado flip-flop D e seu diagrama lógico é apresentado na Fig. 4.34.



(b) Diagrama em bloco

(a) Diagrama lógico

Figura 4.34 Flip-flop tipo D.

EXERCÍCIOS

1) Desenvolva a tabela verdade para as seguintes expressões booleanas:

a) $A \cdot B \cdot C + \overline{A \cdot B \cdot C}$
b) $A \cdot (\overline{C} + B + \overline{D})$
c) $A \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot \overline{C}$
d) $(A + B)\,(\overline{A + C})\,(\overline{\overline{A} \oplus B})$
e) $A \cdot B + A \cdot \overline{B}$
f) $A + \overline{(\overline{B} + A \cdot C)} \oplus \overline{D}$

2) Simplifique as seguintes expressões lógicas:

a) $A \cdot \overline{B} + \overline{B} \cdot A + C \cdot D \cdot E + \overline{C} \cdot D \cdot E + E \cdot \overline{C} \cdot D$

b) $A \cdot B \cdot C \cdot (A \cdot B \cdot C + A \cdot B \cdot C + A \cdot B \cdot C)$

c) $A \cdot B + A \cdot B + A \cdot C + A \cdot C$ d) $(X \cdot Y \cdot Z) \cdot (W \cdot V) \cdot (R \cdot S \cdot T) \cdot (Y \cdot Z \cdot X)$

e) $A \cdot \overline{C} + C \cdot A \cdot B + \overline{C} \cdot A \cdot B + A \cdot C$ f) $(A \cdot C + B \cdot C) \cdot A \cdot (A + A \cdot B) + C \cdot C + A \cdot B$

3) Considere os seguintes valores binários:

A = 1011 B = 1110 C = 0011 D = 1010

Obtenha o valor de X nas seguintes expressões lógicas:

a) $X = A \cdot (B \oplus C)$

b) $X = (\overline{A + B}) \cdot (C \oplus (A + \overline{D}))$

c) $X = B \cdot \overline{C} \cdot A + (\overline{\overline{C} \oplus D})$

d) $X = ((A + \overline{\overline{B} \oplus D}) \cdot (\overline{C} + A) + B) \cdot (\overline{A + B})$

e) $X = A \oplus B + \overline{C} \cdot B + \overline{A}$

4) Use tabelas verdade para mostrar que as expressões abaixo são equivalentes:

a) $(X + Y) + Q$ b) $(X \cdot Y) + Y$

5) Escreva a expressão lógica correspondente a uma porta NAND com 4 entradas.

6) Simplifique as seguintes expressões:

a) $X \cdot Y + X \cdot Y$ b) $(X + Y) \cdot (X + \overline{Y})$ c) $X \cdot Z + X \cdot Y \cdot \overline{Z}$

d) $(A + 1) \cdot (B \cdot 0) + D \cdot D + 1$ e) $(A + 1) \cdot B \cdot \overline{B} + A + C \cdot C + C \cdot 0 + C$

7) Dada a expressão a seguir:

$F = A \cdot B \cdot C + A \cdot B \cdot C + A \cdot B \cdot C$

desenvolva uma expressão equivalente usando apenas operações NAND.

8) O que são portas wired-OR e wired-AND?

9) Qual é a vantagem da tecnologia bipolar sobre a tecnologia FET?

10) Desenhe o diagrama lógico correspondente às seguintes expressões:

a) $X = A \cdot B + (C \cdot D \cdot E)$ b) $X = A + (B + C \cdot D) \cdot (B + A)$

c) $(A + B) \cdot (C + D) \cdot E$ d) $Y = A \cdot B \cdot (C + D) + E$

e) $Y = (A + B) \cdot (C + D) + E$ f) $A + [(B \cdot C) + (D \cdot E)] + F \cdot G + H$

11) Desenvolva a expressão lógica que represente a seguinte afirmação:

"O alarme soará se for recebido um sinal de falha juntamente com um sinal de parada ou um sinal de alerta."

12) Desenvolva a expressão lógica que represente a seguinte afirmação:

"O computador irá funcionar somente se o sinal de energia for recebido ou se for recebido o sinal de força alternativa, mas não se ambos forem recebidos simultaneamente."

13) Projete um circuito lógico que decida o que Bruno deve fazer com relação ao seguinte problema:

- Bruno deverá ir ao jogo de futebol somente se Felipe for à praia e Roberta for estudar.
- Roberta irá estudar se Renata ou Patrícia trouxer o livro.
- Renata concorda em trazer o livro, mas Felipe não quer ir à praia. O que Bruno deverá fazer?

5

Subsistemas de Memória

5.1 INTRODUÇÃO

Conforme já mencionado no Cap. 2, a *memória* é o componente de um sistema de computação cuja função é armazenar as informações que são (ou serão) manipuladas por esse sistema, para que elas (as informações) possam ser prontamente recuperadas, quando necessário.

Conceitualmente, a memória é um componente muito simples: é um "depósito" onde são guardados certos elementos (as informações) para serem usados quando desejado (recuperação da informação armazenada).

No entanto, na prática, em um sistema de computação não é possível construir e utilizar apenas um tipo de memória. Na verdade, a memória de um computador é também em si um sistema, ou melhor, um subsistema, tendo em vista que é constituída de vários componentes (vários tipos diferentes de memória) interligados e integrados, com o objetivo já definido acima: armazenar informações e permitir sua recuperação quando requerido.

A necessidade da existência de vários tipos de memória ocorre em virtude de vários fatores concorrentes. Em primeiro lugar, o aumento, sempre crescente, da velocidade das UCP, muito maior que o *tempo de acesso*[1] da memória, ocasiona atrasos (às vezes, intoleráveis) na transferência de bits entre memória e UCP, e vice-versa. Outro fator relaciona-se com a capacidade de armazenamento de informações que os sistemas de computação precisam ter, cada vez maior, em face do aumento do tamanho dos programas, bem como do aumento do volume dos dados que devem ser armazenados e manipulados nos sistemas atuais. Se existisse apenas um tipo de memória, sua velocidade (tempo de acesso) deveria ser compatível com a da UCP, de modo que esta não ficasse esperando muito tempo por um dado que estivesse sendo transferido da memória.

Explicando melhor, pode-se imaginar um sistema no qual a UCP manipula um dado em 5 nanossegundos, e a memória pode transferir um dado para a UCP em 60 nanossegundos. *Grosso modo* (vamos simplificar bastante o processo para facilitar o entendimento do leitor) pode-se afirmar que a UCP, em cada 60 nanossegundos, trabalharia 5 e ficaria os outros 55 nanossegundos ociosa, acarretando uma baixa produtividade do sistema. Para aumentar esta produtividade pode-se, por exemplo, desenvolver memórias com maior velocidade (isto já vem sendo realizado há muito tempo, como veremos mais adiante). No entanto, tais memórias têm um custo mais elevado que cresce ainda mais quando se sabe que as memórias vêm aumentando de capacidade sistematicamente.

Na realidade, o avanço da tecnologia na construção de processadores e memórias de semicondutores não tem sido uniforme, isto é, o aumento da velocidade das UCP tem sido bem maior que o aumento da velocidade de acesso das memórias. Enquanto a quantidade de instruções executadas por segundo por um processador tem dobrado a cada 18 meses para o mesmo preço, a velocidade de acesso das memórias tem aumentado cerca de 10% ao ano, embora sua capacidade de armazenamento venha quadruplicando a cada 36 meses, para o mesmo preço.

---

[1] A definição completa de tempo de acesso está contida no item 5.2, porém podemos simplificar, estabelecendo que se trata de velocidade da memória.

Em resumo, os dois citados fatores, velocidade e capacidade, indicam a necessidade de se projetar não um único tipo de memória (com elevada velocidade e grande capacidade, mas com custo altíssimo), mas sim um conjunto de memórias com diferentes características, o que leva a uma hierarquia de funcionamento a que denominamos de subsistema de memória. Tal hierarquia será descrita no item 5.2.

A Fig. 5.1 apresenta o esquema conceitual de qualquer tipo de memória, imaginada como um depósito para uso de uma ou mais entidades.

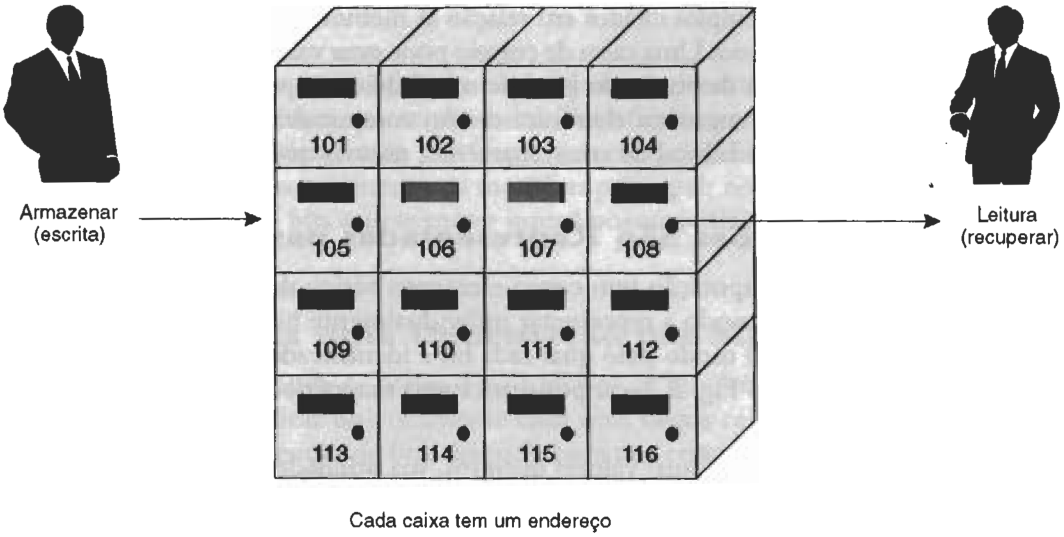


Figura 5.1 Exemplo de um típico depósito que funciona de modo semelhante a uma memória.

Conforme pode ser observado na figura, há duas únicas ações que podem ser realizadas em um depósito (memória). A primeira é a ação de guardar um elemento (ou um grupo de elementos) — em computação, esta ação é genericamente denominada *armazenar* e a operação em si, que é realizada para a consecução desta ação de armazenamento, é chamada de *escrita* ou *gravação* (*write*). A segunda é a ação de recuperação do elemento guardado (ou grupo de elementos) para um uso qualquer — em computação esta ação se denomina *recuperar* (*retrieve*) e a operação para realizá-la chama-se *leitura* (*read*).

Para solidificar mais ainda esses conceitos tão simples e incluir novos conceitos concernentes, pode-se imaginar vários tipos de "depósitos" existentes no nosso dia-a-dia e cujo funcionamento pode ser associado ao das memórias de computadores.

Uma biblioteca,[2] por exemplo, funciona como um "depósito" de elementos (os livros, periódicos etc.). Todo elemento recebido para ser guardado (armazenado) tem que possuir uma identificação (nome do livro ou do periódico, autor etc.) e um código de localização (número da estante, da prateleira etc.) para que seja possível ao funcionário ou usuário encontrar o livro ou periódico quando desejado.

O funcionamento de uma agência do correio para guarda e recuperação de correspondência, ou o modo de manipulação de correspondência em um edifício de apartamentos, é semelhante ao das bibliotecas (apenas no que se refere às operações de armazenamento e leitura) e, de certa forma, ao de uma memória de computador (ou, ainda, ao da própria memória dos seres humanos).

---

[2] Por favor, não pensem os(as) bibliotecários(as) que reduzi a importância e o valor inestimável de uma biblioteca, comparando-a a um simples depósito. A idéia foi tão-somente comparar o modo de funcionamento, não a função.

Em uma biblioteca, o elemento a ser manipulado (a "informação" a que nos referimos antes) é o livro. A ação de armazenamento (que se denomina *escrita* em sistemas de computação), por exemplo, consiste na operação de guardar o livro em uma estante/prateleira previamente identificada como disponível, a qual tem um código de localização, ou seja, prateleira 5 da estante 15 (que é o endereço). Quando alguém deseja um livro emprestado, realiza-se uma operação de "recuperação da informação" (dado o nome do livro, encontra-se a sua localização, que é seu endereço).

Na caixa de correio de um edifício de apartamentos o conceito também é semelhante. A *informação* é a carta. Cada apartamento possui um *endereço* e uma caixa correspondente. A colocação pelo carteiro de uma carta que chega para um determinado apartamento consiste na operação de *armazenamento* (escrita). E a ação do proprietário de apanhar sua correspondência (recuperação ou leitura) é possível através do conhecimento da localização da caixa correspondente.

Há uma pequena diferença nos exemplos citados em relação às memórias de computador, no que se refere à possibilidade de haver um endereço vazio. Uma caixa de correio pode estar vazia porque o responsável retirou toda a correspondência, como também um determinado local de uma biblioteca pode estar vazio porque o livro está emprestado. Isto não acontece com a memória eletrônica de um computador, pois, se ele estiver energizado, a memória conterá sinais elétricos em cada local de armazenamento, mesmo que não seja uma informação útil.

5.1.1 Como as Informações São Representadas nas Memórias

A memória de um sistema de computação tem como elemento básico de armazenamento físico o *bit*. Ou seja, fisicamente ela é construída de modo a representar individualmente bit por bit (seja com seu valor 0 — zero, seja com seu valor 1 — um). O modo pelo qual cada bit é identificado na memória é variado: pode ser um sinal elétrico, como mostrado na Fig. 5.2, ou por um campo magnético ou ainda por presença/ausência de um ponto de luz.

Como um bit pode apenas indicar 2 (dois) valores distintos, sua utilidade individual é bastante restrita. Na prática, precisamos passar para o computador as informações que conhecemos na vida cotidiana, normalmente em forma de caracteres ou símbolos gráficos, que visualmente conseguimos distinguir. Assim, claramente distinguimos o caractere "a" do caractere "b", como também o símbolo matemático "+" do símbolo "(", porque todos eles têm formato visual diferente e o ser humano, através do sentido da visão, consegue distingui-los (é possível, também, para os humanos separar esses símbolos através do sentido do tato, como se faz em braile, para pessoas cegas).

Como computadores não possuem esses sentidos dos humanos, eles conseguem apenas distinguir sinais elétricos diferentes, isto é, se o valor representa 0 (zero) ou 1 (um). Nesse caso, para introduzir todos os símbolos básicos que usamos em nosso dia-a-dia precisaríamos mais do que um bit, visto que, com apenas um bit só poderíamos representar 2 (dois) símbolos distintos. Considerando, por exemplo, que gostaríamos de representar internamente em um computador o nosso alfabeto e demais símbolos gráficos da matemática e afins, precisaríamos criar uma forma de representação interna:

26 letras minúsculas (considerando as letras k, w, x, y)

26 letras maiúsculas

4 símbolos matemáticos (+, −, * , /)

8 sinais de pontuação (. ; , : ( ) − ")

Somente nesse simples exemplo, criamos 64 possibilidades de representação de informações que precisariam ser distinguidas internamente pelo computador, que reconhece apenas 2 (dois) valores diferentes. A solução para esse problema é a definição de um código representativo de cada símbolo, cada um tendo uma mesma quantidade de bits, tantos quantos forem necessários para nos permitir representar todos os símbolos desejados. Nesse exemplo, em que criamos 64 símbolos, precisaríamos definir um código que representasse 64 elementos, ou seja, seriam necessários 6 bits para o código. No item 7.3 o assunto é abordado com mais detalhe e o Apêndice B apresenta alguns códigos completos para a representação de dados.

Por isso, os sistemas de computação costumam grupar uma determinada quantidade de bits, identificando este grupo como uma unidade de armazenamento, denominada *célula*.

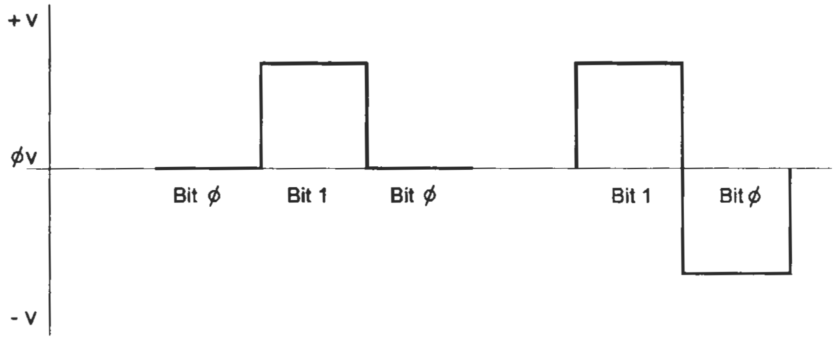


Figura 5.2 Representação de 1 bit.

Uma *célula* é, então, um grupo de bits tratado em conjunto pelo sistema, isto é, esse grupo é movido em bloco como se fosse um único elemento, sendo assim identificado para efeitos de armazenamento e transferência, como uma unidade. Mais adiante, conforme veremos, o termo *célula* costuma ser utilizado apenas para identificar a unidade de armazenamento da memória principal; nos demais tipos de memória, a unidade de armazenamento (grupo de bits que se move junto) possui outras denominações (bloco, setor, cluster etc.).

5.1.2 Como se Localiza uma Informação nas Memórias

Como uma memória é constituída de vários desses grupos de bits (célula, bloco etc.) é necessário que seja definido um método para identificar univocamente cada uma dessas células (ou blocos), de modo que possa ser distintamente identificado o grupo de bits desejado para um certo processamento.

Trata-se de um processo semelhante a vários outros encontrados em nosso dia-a-dia. Por exemplo, em uma rua pode-se construir 100 casas rigorosamente iguais, de tal forma que não se pode visualmente distinguir uma da outra. No entanto, o carteiro não erra ao entregar diariamente a correspondência, pois cada casa possui uma forma única de identificação, um número fixado em sua porta ou fachada. Este número, diferente e único para cada casa, é denominado *endereço*.

De modo análogo, em um sistema de computação as células (ou grupos de bits que se movem junto) são identificadas, uma a uma, por um número, também denominado *endereço*. Na Fig. 5.1 podemos observar que cada caixa possui um número identificador, 101, 102, 103, 104, ... 116, que é seu endereço.

Em resumo, cada célula da memória principal ou cada grupo de bits (bloco, setor etc.) em um sistema de computação é identificado na sua fabricação por um número denominado endereço. A memória é organizada, então, em grupos de bits, seqüencialmente dispostos, a partir do grupo (célula, bloco, setor etc.) de endereço 0 (zero) até o último grupo, de endereço (N – 1), sendo N a quantidade total de grupos. O sistema de controle das memórias é construído de modo a localizar um certo grupo de bits a partir do seu endereço, conforme veremos mais adiante.

As Figs. 5.7 a 5.10 mostram memórias com a disposição de suas células e respectivos endereços.

É importante enfatizar (e isso será repetido em outras partes deste livro) que as memórias são constituídas de elementos físicos (seu conteúdo) que, de diferentes formas (elétrica, magnética, ótica), representam os dados que desejamos armazenar e manipular. Os endereços de cada grupo de bits (célula, bloco, setor etc.) **não são** fisicamente representados em qualquer lugar do sistema. Didaticamente insere-se, ao lado de cada célula, por exemplo, o seu endereço de modo que o leitor possa identificar a célula e acompanhar melhor a explicação, mas o valor do endereço não está presente fisicamente naquela posição.

5.1.3 Operações Realizadas em uma Memória

Já foi mencionado que uma memória é um dispositivo de armazenamento (depósito) de informações. Quando se organiza um depósito, tem-se por objetivo permitir que elementos (objetos, no caso de uma fábrica, ou informações, no caso das memórias) possam ser guardados (armazenados) de uma forma organizada que possibilite sua identificação e localização, quando se deseja recuperá-los (apanhar para uso). Pode-se, então, realizar duas ações distintas em um depósito:

- guardar o elemento (armazenar); e
- retirar o elemento (recuperar).

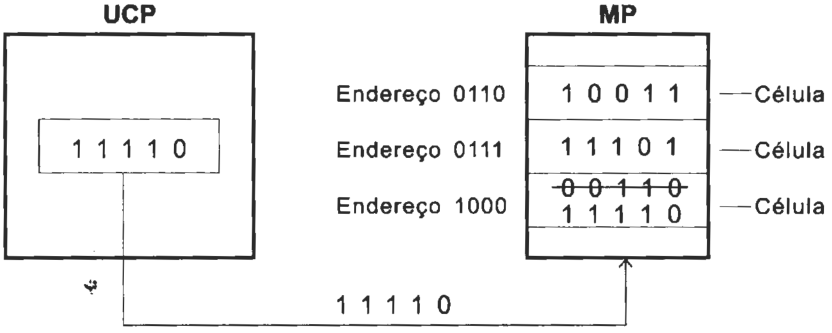
Também em uma memória pode-se realizar essas mesmas duas ações (ou operações), as quais, nesse caso, são denominadas:

- escrita ou gravação ou armazenamento (*write* ou *record*); e
- leitura ou recuperação (*read* ou *retrieve*).

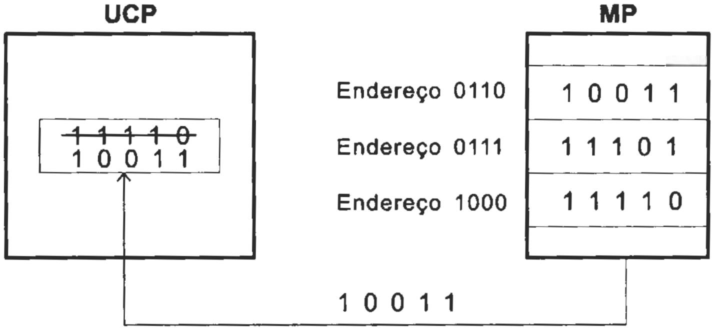
Ambas as operações são possíveis graças à técnica utilizada para identificar cada elemento (ou grupo de bits) por um número, seu endereço, que permite identificar o local de armazenamento (para escrita) ou de recuperação (para leitura).

A operação de escrita é naturalmente destrutiva, ou seja, ao armazenar-se um dado em uma célula o conteúdo anterior é destruído, visto que os bits que chegam são gravados por cima dos que estavam no local. O processo é semelhante ao realizado para gravar uma música em uma fita cassete — a música que vai sendo gravada apaga a anterior. A Fig. 5.3(a) mostra um exemplo de operação de escrita, vista em detalhes no item 5.3.3.2.

A operação de leitura, cujo exemplo é apresentado na Fig. 5.3(b), não deve ser destrutiva. Ela é, na realidade, uma ação de copiar um valor (dado ou informação) em outro local, permanecendo o mesmo valor no local de origem.



(a) Operação de escrita — O valor 11110 é transferido (uma cópia) da UCP—para a MP e armazenado na célula de endereço 1000, apagando o conteúdo anterior (00110).



(b) Operação de leitura — O valor 10011, armazenado no endereço da MP 0110 é transferido (cópia) para a UCP, apagando o valor anterior (11110) e armazenando no mesmo local.

Figura 5.3 Operação de leitura e escrita na MP.

5.2 HIERARQUIA DE MEMÓRIA

No item anterior mencionamos a necessidade de se projetar sistemas de computação que adotassem uma memória constituída de um conjunto de diferentes tipos, organizados de forma hierárquica.

Na realidade, há muitas memórias no computador, as quais se interligam de forma bem estruturada, constituindo um sistema em si, fazendo parte do sistema global de computação, que pode ser denominado *subsistema de memória.*

Esse subsistema é projetado de modo que seus componentes sejam organizados hierarquicamente, conforme mostrado na estrutura em forma de pirâmide da Fig. 5.4.

A pirâmide em questão é projetada com uma base larga, que simboliza a elevada capacidade, o tempo de uso e o custo do componente que a representa.

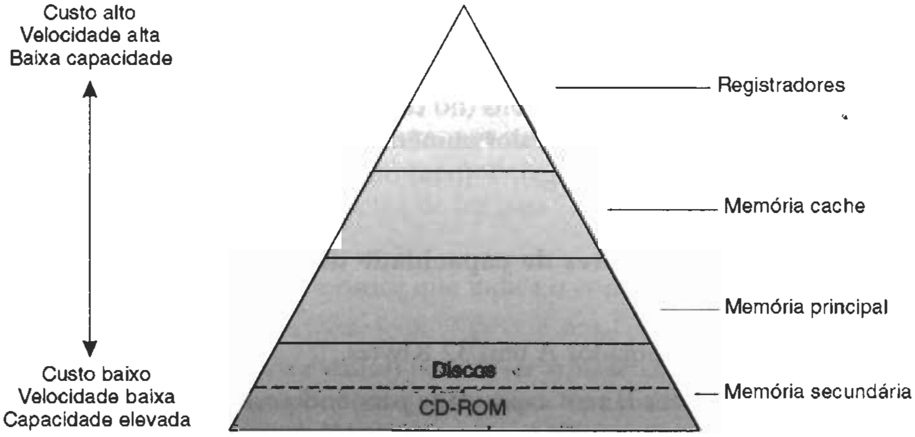


Figura 5.4 Hierarquia de memória.

A variação crescente dos valores de certos parâmetros que caracterizam um tipo de memória pode ser mostrada no formato inclinado de uma pirâmide.

A seguir serão definidos os principais parâmetros para análise das características de cada tipo de memória componente da hierarquia apresentada na Fig. 5.4. O valor maior (base) ou menor (pico) de algum parâmetro foi o motivo de utilizarmos uma pirâmide para representar a hierarquia do sistema de memória de um computador.

Tempo de acesso — indica quanto tempo a memória gasta para colocar uma informação na barra de dados após uma determinada posição ter sido endereçada. Isto é, o período de tempo decorrido desde o instante em que foi iniciada a operação de acesso (quando a origem — em geral é a UCP — passa o endereço de acesso para o sistema de memória) até que a informação requerida (instrução ou dado) tenha sido efetivamente transferida. É um dos parâmetros que pode medir o desempenho da memória. Pode ser chamado de *tempo de acesso* para leitura ou simplesmente tempo de leitura.

O valor do tempo de acesso de uma memória é dependente da sua tecnologia de construção e da velocidade de seus circuitos. Ele varia bastante para cada tipo, de alguns poucos nanossegundos, no caso de memórias tipo RAM (DRAM, SRAM etc. — são descritas mais adiante) até dezenas ou centenas de milissegundos, no caso de memória secundária (discos magnéticos, CD-ROMs e fitas).

A descrição desses tipos, seu funcionamento e características básicas serão apresentados mais adiante (neste capítulo será tratada a memória principal, RAM/ROM e memória cache, ficando a memória secundária para ser discutida no Cap. 10 Entrada e Saída (E/S)).

Deve ser mencionado ainda que o tempo de acesso das memórias eletrônicas (do tipo RAM, ROM etc.) é o mesmo, independentemente da distância física entre o local de um acesso e o local do próximo acesso, ao passo que, no caso de dispositivos eletromecânicos (discos, CD-ROMs etc.), o tempo de acesso varia conforme a distância física entre dois acessos consecutivos.

Outro parâmetro (utilizado apenas em memórias eletrônicas) é o chamado ciclo de tempo do sistema de memória (*memory system's cycle time*) ou simplesmente *ciclo de memória,* que é o período de tempo decorrido entre duas operações sucessivas de acesso à memória, sejam de escrita ou de leitura. Esse tempo depende de outros fatores relacionados aos tempos de funcionamento do sistema. Esses outros fatores podem, em certas memórias, impedir, por um pequeno intervalo de tempo, o uso do sistema de memória para um novo acesso, logo após a conclusão do acesso anterior. Nesses casos, o ciclo de memória compreende o tempo de acesso mais um certo tempo para essas outras atividades, a serem descritas mais adiante. Outras memórias não requerem esse tempo adicional entre acessos e, portanto, o ciclo de memória é igual ao tempo de acesso.

O ciclo de memória é usualmente empregado como elemento de medida de desempenho das memórias eletrônicas, sendo indicado nos manuais e demais documentos descritivos das características de um dado tipo de memória.

Capacidade é a quantidade de informação que pode ser armazenada em uma memória. A unidade de medida mais comum é o *byte,* embora também possam ser usadas outras unidades como *células* (no caso de memória principal ou cache), setores (no caso de discos) e bits (no caso de registradores). Dependendo do tamanho[3] da memória, isto é, de sua capacidade, indica-se o valor numérico total de elementos de forma simplificada, através da inclusão de K (quilo), M (mega), G (giga), T (tera) ou P (peta) (ver item 2.2.1).

Exemplos de nomenclatura para valores de capacidade de memórias:

- O registrador R1 tem 32 bits.
- A memória ROM do microcomputador A tem 32 Kbytes.
- A memória RAM do computador B tem capacidade para endereçar 128M células.
- O disco C tem capacidade para armazenar 8,2 Gbytes.
- O CD-ROM E tem capacidade de armazenamento igual a 650 Mbytes.

Volatilidade — memórias podem ser do tipo volátil ou não-volátil. Uma memória não-volátil é a que retém a informação armazenada quando a energia elétrica é desligada. Memória volátil é aquela que perde a informação armazenada quando o equipamento é desligado (interrupção de alimentação elétrica ou desligamento da chave ON/OFF do equipamento).

Uma vez que um processador nada pode fazer sem instruções que indiquem a próxima operação a ser realizada, é óbvio que todo sistema de computação deve possuir alguma quantidade de memória não-volátil. Isto é, ele deve possuir, pelo menos, algumas instruções armazenadas em memória não-volátil para serem executadas inicialmente, sempre que se ligar o computador.

Registradores são memória do tipo volátil, como também memórias de semicondutores, do tipo RAM. Memórias magnéticas e óticas, como discos e fitas, e memórias de semicondutores do tipo ROM, EPROM etc. são também do tipo *não-volátil.* É possível manter a energia em uma memória originalmente não-volátil, com o emprego de baterias. Mais adiante, este parâmetro será analisado e exemplificado com um pouco mais de detalhe (ver item 5.6).

Tecnologia de fabricação — ao longo do tempo, diversas tecnologias vêm sendo desenvolvidas para a fabricação de memórias. Atualmente, algumas dessas tecnologias já são obsoletas, como as memórias de núcleo de ferrite (magnéticos), e outras ainda não têm uma aplicação comercial ampla, como as memórias de bolha. Algumas das tecnologias mais conhecidas e utilizadas são:

---

[3] Em computação, costuma-se usar o termo tamanho para indicar a quantidade de informação (bits, bytes etc.) e não para indicar a grandeza física do elemento, como se faz na vida cotidiana. Por exemplo, o tamanho da barra de dados de um certo sistema é de 64 bits, significando que a referida barra de dados possui 64 fios colocados juntos, cada um permitindo a passagem de um sinal elétrico, correspondente ao valor de um bit.

a) *Memórias de semicondutores* — são dispositivos fabricados com circuitos eletrônicos e baseados em semicondutores. São rápidas e relativamente caras, se comparadas com outros tipos. Dentro desta categoria geral há várias tecnologias específicas, cada uma com suas vantagens, desvantagens, velocidade, custo etc., as quais serão mais detalhadamente descritas no item 5.6.

Registradores, memória principal e memória cache são exemplos de memórias de semicondutores ou, mais simplesmente, memórias eletrônicas.

b) *Memórias de meio magnético* — são dispositivos, como os disquetes e discos rígidos (*hard disks*), fabricados de modo a armazenar informações sob a forma de campos magnéticos. Eles possuem características magnéticas semelhantes às das fitas cassetes de som, que são memórias não-voláteis. Devido à natureza eletromecânica de seus componentes e à tecnologia de construção em comparação com memórias de semicondutores, esse tipo de memória é mais barato e permite, assim, o armazenamento de grande quantidade de informação. O método de acesso às informações armazenadas em discos e fitas é diferente (ver itens 10.3.4 e 10.3.5), resultando em tempos de acesso diversos (por possuírem acesso direto, discos são mais rápidos do que fitas, que operam com acesso seqüencial). O Cap. 10 (Entrada e Saída) descreve, com mais detalhe, a organização, as tecnologias e outras características dos discos e fitas magnéticas.

c) *Memórias de meio ótico* — são dispositivos, do tipo CD-ROM, capazes de armazenar cerca de 650 Mbytes de informação. É permitida apenas a sua leitura, por isso o nome ROM após o CD (já existem no mercado dispositivos de armazenamento ótico que podem regravar dados — são os CD-RW (readable/writeble). Tais dispositivos utilizam um feixe de luz para "marcar" o valor (0 ou 1) de cada dado em sua superfície.

Temporariedade — trata-se de uma característica que indica o conceito de tempo de permanência da informação em um dado tipo de memória.

Por exemplo, informações (programas e dados) podem ser armazenadas em discos ou disquetes e lá permanecerem armazenadas indefinidamente (por "indefinidamente" entende-se um considerável período de tempo de muitos anos, mas há sempre a possibilidade de perda de magnetismo com o passar do tempo). Pode-se, então, definir esse tipo de memória como *permanente*. Ao contrário dos registradores que armazenam um dado por um tempo extremamente curto (nanossegundos) o suficiente para o dado ser, em seguida, transferido para a UAL. Os registradores podem, às vezes, reter o dado armazenado para posterior processamento pela UAL, mas, mesmo assim, esta retenção não dura mais do que o tempo de execução do programa (na hipótese de maior permanência) ou de parte dele. É uma memória do tipo *transitória*. Outros exemplos de memórias de permanência transitória de dados são a memória cache e a memória principal, embora os dados nelas permaneçam armazenados por mais tempo do que nos registradores (tempo de duração da execução de um programa — que pode ser de uns poucos segundos ou até mesmo de algumas horas).

Custo — o custo de fabricação de uma memória é bastante variado em função de diversos fatores, entre os quais se pode mencionar principalmente a tecnologia de fabricação, que redunda em maior ou menor tempo de acesso, ciclo de memória, quantidade de bits em certo espaço físico e outros. Uma boa unidade de medida de custo é o preço por byte armazenado, em vez do custo total da memória em si. Isso porque, devido às diferentes capacidades, seria irreal considerar, para comparação, o custo pelo preço da memória em si, naturalmente diferente, e não da unidade de armazenamento (o byte), igual para todos os tipos.

Em outras palavras, um disco rígido de microcomputador pode armazenar cerca de 10GB e custar, no mercado, em torno de U$150,00, o que indica um custo de U$0,000001 centavo por byte, enquanto uma memória do tipo de semicondutor, dinâmica (ver item 5.7), pode custar cerca de U$15,00 por Mbyte, adquirindo-se, então, 32 MB por U$45,00 (o preço de 1 byte seria de 0,0001 centavo, muito mais caro que o byte armazenado no disco. Não há comparação possível entre o valor dos 32 MB de RAM e os 10GB do disco se analisarmos apenas o dispositivo como um todo (o disco seria mais caro).

5.2.1 Registradores

Em um sistema de computação, a destinação final do conteúdo de qualquer tipo de memória é o processador (a UCP). Isto é, o objetivo final de cada uma das memórias (ou do subsistema de memória) é armazenar informações destinadas a serem, em algum momento, utilizadas pelo processador. Ele é o responsável pela execução das *instruções,* pela manipulação dos *dados* e pela produção dos resultados das operações.

As ações operativas do processador são realizadas (ver Cap. 6) nas suas unidades funcionais: na unidade aritmética e lógica — UAL (ALU — Arithmetic and Logic Unit), na unidade de ponto flutuante — UFP (Float Point Unit — FPU) ou talvez em uma unidade de processamento vetorial. No entanto, antes que a instrução seja interpretada e os dispositivos da UCP sejam acionados, o processador necessita buscar a instrução de onde ela estiver armazenada (memória cache ou principal) e armazená-la em seu próprio interior, em um dispositivo de memória denominado *registrador de instrução.*

Em seguida a este armazenamento da instrução, o processador deverá, na maioria das vezes, buscar dados da memória (cache, principal ou mesmo de unidades de disco em fita) para serem manipulados na UAL. Esses dados também precisam ser armazenados em algum local da UCP até serem efetivamente utilizados. Os resultados de um processamento (de uma soma, subtração, operação lógica etc.) também precisam, às vezes, ser guardados temporariamente na UCP, ou para serem novamente manipulados na UAL por uma outra instrução, ou para serem transferidos para uma memória externa à UCP. Esses dados são armazenados na UCP em pequenas unidades de memória, denominadas *registradores.*

Um registrador é, portanto, o elemento superior da pirâmide de memória (ver Fig. 5.2), por possuir a maior velocidade de transferência dentro do sistema (menor tempo de acesso), menor capacidade de armazenamento e maior custo.

Analisando os diversos parâmetros que caracterizam as memórias, descritos no item anterior, temos:

Tempo de acesso — 1 ciclo de memória — por serem construídos com a mesma tecnologia da UCP, estes dispositivos possuem o menor tempo de acesso/ciclo de memória do sistema (neste caso, não é aplicável distinguir-se tempo de acesso e ciclo de memória, por serem sempre iguais), algo em torno de 1 a 5 nanossegundos.[4]

Capacidade — os registradores são fabricados com capacidade de armazenar um único dado, uma única instrução ou até mesmo um único endereço. Dessa forma, a quantidade de bits de cada um é de uns poucos bits (de 8 a 64), dependendo do tipo de processador e, dentro deste, da aplicação dada ao registrador em si. Registradores de dados têm, em geral, o tamanho definido pelo fabricante para a palavra do processador, tamanho diferente dos registradores usados exclusivamente para armazenar endereços (quando há registradores com esta função específica no processador). Por exemplo, o processador Intel Pentium, cuja palavra é de 32 bits (ver Cap. 6), tem registradores também de 32 bits, inclusive registradores de endereços (os números que indicam os endereços de célula de memória principal do processador têm 32 bits). Já o processador Motorola 68000 tem registradores de dados de 32 bits (palavra de 32 bits) e registrador de endereços de 24 bits, enquanto o processador Alpha possui registradores de dados de 64 bits, assim como os Pentium mais avançados.

Volatilidade — registradores são memórias de semicondutores e, portanto, necessitam de energia elétrica para funcionar (há exceções, as quais serão analisadas no item 5.3.5). Assim, registradores são memórias voláteis. Para a UCP funcionar sem interrupção, mesmo quando eventualmente a energia elétrica para o computador é interrompida, é necessário que o sistema de computação seja ligado a um dispositivo de alimentação elétrica denominado *no-break*, que é constituído de bateria ou gerador de corrente e um conversor AC/DC.

Tecnologia — conforme mencionado no tópico anterior, os registradores são memórias de semicondutores, sendo fabricados com tecnologia igual à dos demais circuitos da UCP, visto que eles se encontram inseridos em seu interior. No entanto, há diversos modelos de tecnologia de fabricação de semicondutores, uns com tempo de acesso maior que outros, custos e capacidade de armazenamento, no mesmo espaço físico, diferentes. Tecnologia bipolar e MOS (Metal Oxide Semicondutor) são comuns na fabricação de registradores, sendo descritas no item 5.6.

---

[4] É importante mencionar que os números que representam o desempenho de um computador são bastante variáveis, visto que a tecnologia avança em enorme velocidade (maior do que a rapidez com que eu atualizo as versões deste livro). Assim, citar 1 a 5 ns neste instante pode significar um valor conservador quando este livro estiver sendo lido, por estarem já ultrapassados ou desatualizados.

Temporariedade — os registradores são memórias auxiliares internas à UCP e, portanto, tendem a armazenar informação (dados ou instruções) por muito pouco tempo. Acumuladores ou registradores de dados armazenam os dados apenas pelo tempo necessário para sua utilização na UAL.

Custo — devido à tecnologia mais avançada de sua fabricação, os registradores encontram-se no topo da pirâmide em termos de custos, sendo os dispositivos de maior custo entre os diversos tipos de memória.

5.2.2 Memória Cache

Na pirâmide de memória, abaixo dos registradores, encontra-se a memória cache. Em sistemas de computação mais antigos, a pirâmide não possuía memória cache e, desse modo, os registradores eram ligados diretamente à memória principal. A Fig. 5.5 mostra uma pastilha (*chip*) de memória cache (cache externa).

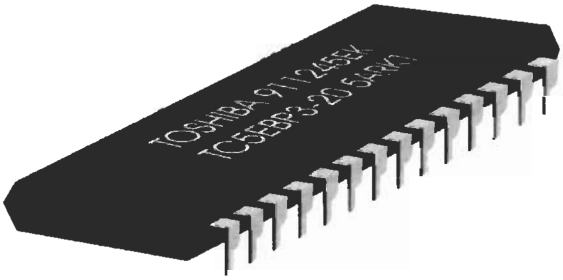


Figura 5.5 Exemplo de memória cache (cache externo).

Em toda execução de uma instrução, a UCP acessa a memória principal (sem cache), pelo menos uma vez, para buscar a instrução (uma cópia dela) e transferi-la para um dos registradores da UCP (ver Cap. 6). Muitas instruções requerem outros acessos à memória, seja para a transferência de dados para a UCP (que serão processados na UAL), seja para a transferência do resultado de uma operação da UCP para a memória.

Em resumo, para a realização do ciclo de uma instrução há sempre a necessidade de ser realizado um ou mais ciclos de memória (no Cap. 6 será descrito com detalhes o funcionamento dos ciclos de instrução).

Considerando-se que um ciclo de memória é atualmente bem mais demorado do que o período de tempo que a UCP gasta para realizar uma operação na UAL, fica claro que a duração da execução de um ciclo de instrução é bastante afetada pela demora dos ciclos de memória.

Desde há muito, então, esta interface entre o processador e a memória principal vem sendo um ponto frágil no que se refere à performance do sistema.

Na tentativa de melhorar o desempenho dos sistemas de computação, os projetistas das UCP vêm constantemente obtendo velocidades cada vez maiores nas operações dessas unidades, o que não está acontecendo na mesma proporção com o aperfeiçoamento tecnológico das memórias utilizadas como memória principal. Assim, atualmente a diferença de velocidade entre UCP e memória principal é talvez maior do que já foi no passado.

Na busca de uma solução para este problema (o gargalo de congestionamento na comunicação UCP/MP que degrada o desempenho dos sistemas), foi desenvolvida uma técnica que consiste na inclusão de um dispositivo de memória entre UCP e MP, denominado memória CACHE, cuja função é acelerar a velocidade de transferência das informações entre UCP e MP e, com isso, aumentar o desempenho dos sistemas de computação.

Para tanto, esse tipo de memória é fabricado com tecnologia semelhante à da UCP e, em conseqüência, possui tempos de acesso compatíveis com a mesma, resultando numa considerável redução da espera da UCP para receber dados e instruções da cache, ao contrário do que acontece em sistemas sem cache (conceitos e tecnologia de fabricação de memórias cache, bem como seu funcionamento, são detalhadamente apresentados no item 5.5).

Atualmente há diversos tipos de memória cache, utilizados em sistemas de computação modernos: RAM cache ou cache para a memória principal e cache para disco. Trataremos aqui das características das memórias cache para memória principal, deixando para o item 5.5 descrever as características das memórias cache para disco.

Além disso, deve-se mencionar que as memórias RAM cache podem ser inseridas em dois (ou até três) níveis. O primeiro nível é denominado L1 (Level 1 — nível 1), uma memória cache inserida internamente no processador, isto é, é encapsulada na mesma pastilha, enquanto a de nível 2, L2 ou cache externa (ou ainda cache secundária) consiste em uma pastilha (*chip*) separada e própria, instalada na placa-mãe do computador. No item 5.5 descreveremos esses tipos um pouco mais detalhadamente.

A memória cache apresenta as seguintes características para os já referidos parâmetros:

Tempo de acesso — ciclo de memória — sendo as memórias de semicondutores, fabricadas com tecnologia e recursos para prover menores ciclos de memória do que as memórias RAM comuns (memória principal do tipo dinâmica, a ser explicada nos itens 5.3 e 5.7), elas possuem velocidade de transferência tal que lhes garante tempos de acesso menores que 5 a 7 ns (nanossegundos), sendo por esta razão situadas, na pirâmide, logo abaixo dos registradores.

Capacidade — tendo em vista que a UCP acessa primeiramente a memória cache, para buscar a informação requerida (a próxima instrução ou dados requeridos pela instrução em execução), é importante que a referida memória tenha capacidade adequada para armazenar uma apreciável quantidade de informações, uma vez que, se essas informações não foram encontradas na cache, então o sistema deverá sofrer um atraso para que as mesmas sejam transferidas da memória principal para a cache.

Por outro lado, uma grande capacidade implicará certamente elevação de seu custo, muitas vezes inaceitável para compor o preço total do sistema.

Dessa forma, deve-se conciliar o compromisso de uma apreciável capacidade com a não-elevação demasiada de seu preço.

Valores típicos de memória cache oscilam entre 64K e 2MB de cache secundária, ou L2, e da ordem de 16K ou mais para a cache primária, ou L1, internas ao processador.

Volatilidade — a exemplo dos registradores, memórias cache são dispositivos construídos com circuitos eletrônicos, requerendo, por isso, energia elétrica para seu funcionamento. São, desse modo, dispositivos voláteis.

Tecnologia — memórias cache são fabricadas com circuitos eletrônicos de alta velocidade para atingirem sua finalidade. Em geral, são memórias estáticas, denominadas SRAM (ver item 5.3.5).

Temporariedade — o tempo de permanência de uma instrução ou dado nas memórias cache é relativamente pequeno, menor que a duração da execução do programa ao qual a referida instrução ou dado pertence. Isto porque, devido a seu tamanho não ser grande e ser utilizada por todos os programas em execução, há necessidade de alteração periódica da informação armazenada para permitir a entrada de novas informações. Embora a transitoriedade das informações na cache seja uma realidade, o período efetivo de permanência de um dado ou instrução é dependente do tipo de política de substituição de informação na cache (ver item 5.5).

Custo — o custo de fabricação das memórias cache é alto. O valor por byte está situado entre o dos registradores, que são os mais caros, e o da memória principal, mais barata. Memórias cache internas à UCP ainda são mais caras do que as externas.

5.2.3 Memória Principal

Uma das principais características definidas no projeto de arquitetura do sistema de von Neumann, o qual se constitui na primeira geração dos computadores, consistia no fato de ser uma máquina "de programa armazenado". O fato de as instruções, uma após a outra, poderem ser imediatamente acessadas pela UCP é que garante o automatismo do sistema e aumenta a velocidade de execução dos programas (por se tratar de uma máquina executando ações sucessivas, sem intervalos e sem cansar, como não acontece com os seres humanos).

A UCP pode acessar imediatamente uma instrução após a outra porque elas estão armazenadas internamente no computador. Esta é a importância da memória.

E, desde o princípio, a memória especificada para armazenar o programa (e os seus dados) a ser executado é a memória que atualmente chamamos de principal (ou memória real), para distingui-la da memória de discos e fitas (memória secundária).

A memória principal é, então, a memória básica de um sistema de computação desde seus primórdios. É o dispositivo onde o programa (e seus dados) que vai ser executado é armazenado para que a UCP busque instrução por instrução, para executá-las.

Seus parâmetros possuem as seguintes características:

Tempo de acesso — ciclo de memória — a memória principal é construída com elementos cuja velocidade operacional se situa abaixo das memórias cache, embora sejam muito mais rápidas que a memória secundária. Nas gerações iniciais de computadores o tipo mais comum de memória principal era uma matriz de pequenos núcleos magnéticos, os quais armazenavam o valor 1 ou o valor 0 de bit, conforme a adição do campo magnético armazenado. Essas memórias possuíam baixa velocidade, a qual foi substancialmente elevada com o surgimento das memórias de semicondutores. Atualmente, as memórias desse tipo possuem tempo de acesso entre 7 e 15 ns.

Capacidade — em geral, a capacidade da memória principal é bem maior que a da memória cache. Enquanto esta oscila atualmente entre 64K e 2 Mbytes, valores típicos de memória principal para microcomputadores estão na faixa de dezenas de Mbytes (1000K), pois raramente vai se adquirir, nos dias de hoje, um microcomputador que não possua algo em torno de 32MB de memória principal. Já se pode instalar estes computadores com 64 até 512 Mbytes em um computador, embora eles possam endereçar memórias de 4 Gbytes (gigabytes).

Volatilidade — sendo atualmente construído com semicondutores e circuitos eletrônicos correlatos, este tipo de memória também é volátil, tal como acontece com os registradores e a memória cache. No entanto, há normalmente uma pequena quantidade de memória não-volátil fazendo parte da memória principal, a qual serve para armazenar pequena quantidade de instruções que são executadas sempre que o computador é ligado.

Tecnologia — conforme já mencionado, nos primeiros sistemas usavam-se núcleos de ferrite (processo magnético) para armazenar os bits na memória principal, até que foram substituídos pela tecnologia de semicondutores. Os circuitos que representam os bits nas memórias atuais possuem uma tecnologia bem mais avançada que seus predecessores de ferrite e, portanto, têm velocidade mais elevada de transferência, garantindo baixos tempos de acesso em comparação com o modelo anterior. São, porém, elementos mais lentos do que aqueles que constituem as memórias cache. Na maioria dos sistemas atuais esta tecnologia produz memória com elementos dinâmicos (DRAM), como será mostrado nos itens 5.3.5 e 5.6.

Temporariedade — para que um programa seja executado é necessário que ele esteja armazenado na memória principal juntamente com seus dados. Atualmente esta afirmação é parcialmente verdadeira, visto que não é mais necessário que o programa completo (todas as instruções) esteja na MP. Neste caso, é obrigatório apenas o armazenamento, na MP, da instrução que será acessada pela UCP (na prática, não se usa somente a *instrução* que será executada, mas sim esta e um grupo de outras). Não importa, contudo, se é o programa todo, ou parte dele, que deve estar armazenado na MP para ser utilizado pela UCP. Fica claro que, em qualquer circunstância, as instruções e os dados permanecem temporariamente na MP, enquanto durar a execução do programa (ou até menos tempo). Esta temporariedade é bastante variável, dependendo de diversas circunstâncias, como, por exemplo, o tamanho do programa e sua duração, a quantidade de programas que estão sendo processados juntos, e outras mais. No entanto, a transitoriedade com que as informações permanecem armazenadas na MP é, em geral, mais duradoura que na memória cache ou nos registradores, embora mais lenta que na memória secundária.

Custo — memórias dinâmicas usadas como memória principal têm um custo mais baixo que o custo das memórias cache, por isso podem ser vendidos computadores com uma quantidade apreciável de MP (com 32 MB, 64MB e até 128MB) sem que seu preço seja inaceitável.

5.2.4 Memória Secundária

Na base da pirâmide que representa a hierarquia de memória em um sistema de computação encontra-se um tipo de memória com maior capacidade de armazenamento do que os outros tipos já descritos, menor custo por byte armazenado e com tempos de acesso também superiores aos outros tipos. Esta memória, denominada memória secundária, memória auxiliar ou memória de massa, tem por objetivo garantir um armazenamento mais permanente aos dados e programas do usuário, razão por que deve naturalmente possuir maior capacidade que a memória principal.

A memória secundária de um sistema de computação pode ser constituída por diferentes tipos de dispositivos, alguns diretamente ligados ao sistema para acesso imediato (discos rígidos, por exemplo) e outros que podem ser conectados quando desejado (como disquetes, fitas de armazenamento, CD-ROM etc.), cuja informação armazenada se torna diretamente conectada e disponível quando o específico disquete ou fita estiver inserido no elemento de leitura/escrita (*drive* ou acionador), enquanto os demais ficam disponíveis (*off-line*) para acesso manual pelo usuário.

Uma das principais características dos dispositivos que constituem a memória secundária é sua não-volatilidade.

A análise de seus parâmetros (considerando uma média entre os diversos equipamentos) conduz às seguintes observações:

Tempo de acesso — ciclo de memória — conforme será mostrado no Cap. 10, os dispositivos que podem se constituir em elemento de armazenamento secundário ou auxiliar em um sistema de computação são, em geral, dispositivos eletromecânicos e não circuitos puramente eletrônicos, como é o caso de registradores, memória cache e memória principal. Por essa razão, aqueles dispositivos possuem tempo de acesso maiores.

Por mais tecnologicamente avançado que seja, um mecanismo que movimenta fisicamente um braço mecânico (no caso dos discos rígidos) requer tempos sempre muito maiores que o tempo requerido para um sinal elétrico passar por um condutor, já que este não possui elementos mecânicos e caminha com a velocidade da luz.

Tempos de acesso típicos para discos rígidos estão atualmente na faixa de 8 a 15 milissegundos (já estão sendo fabricados discos com tempos ainda menores, mas sempre da ordem de milissegundos, muito acima dos nanossegundos das outras memórias). Discos do tipo CD-ROM trabalham com tempos de acesso ainda maiores, na faixa de 120 a 300 ns, enquanto as fitas magnéticas são ainda mais lentas, podendo ler um arquivo em tempos da ordem de segundos.

Capacidade — uma das características que coloca a memória secundária na base da pirâmide é justamente sua grande capacidade de armazenamento, a qual também varia consideravelmente dependendo do tipo de dispositivo utilizado.

Discos rígidos de microcomputadores podem, atualmente, ser encontrados com capacidades variando entre 2 e 50 Gbytes. Sistemas de grande porte já há muito tempo possuem discos rígidos com dezenas de Gbytes de capacidade de armazenamento. Os demais dispositivos, como CD-ROM (com capacidade da ordem de 600MB ou mais para cada disco, podendo o usuário possuir tantos quantos quiser e puder pagar), fitas magnéticas (a capacidade depende do comprimento da fita e da densidade de gravação), discos óticos etc., serão analisados com mais detalhes no Cap. 10.

Volatilidade — como estes dispositivos armazenam as informações de forma magnética ou ótica, elas não se perdem nem desaparecem quando não há alimentação de energia elétrica. Trata-se, pois, de elementos úteis para guardar os programas e dados de forma permanente.

Tecnologia — este parâmetro possui uma variedade imensa de tipos, visto que, para cada dispositivo entre os já mencionados (discos, disquetes, fitas, discos óticos, CD-ROM), há diferentes tecnologias de fabricação atualmente em uso, o que dificulta a sua descrição neste item. No Cap. 10 serão apresentados mais detalhes de cada dispositivo, incluindo-se algumas das tecnologias em uso.

Temporariedade — conforme já mencionado em Volatilidade, a memória secundária é um componente (vendo-se os diversos dispositivos como um todo) de armazenamento com caráter permanente ou, pelo menos, de longo período de armazenamento. Ela serve, então, para armazenar programas e dados que não estão sendo requeridos imediatamente e que exigem também grande espaço de armazenamento devido à sua natural quantidade.

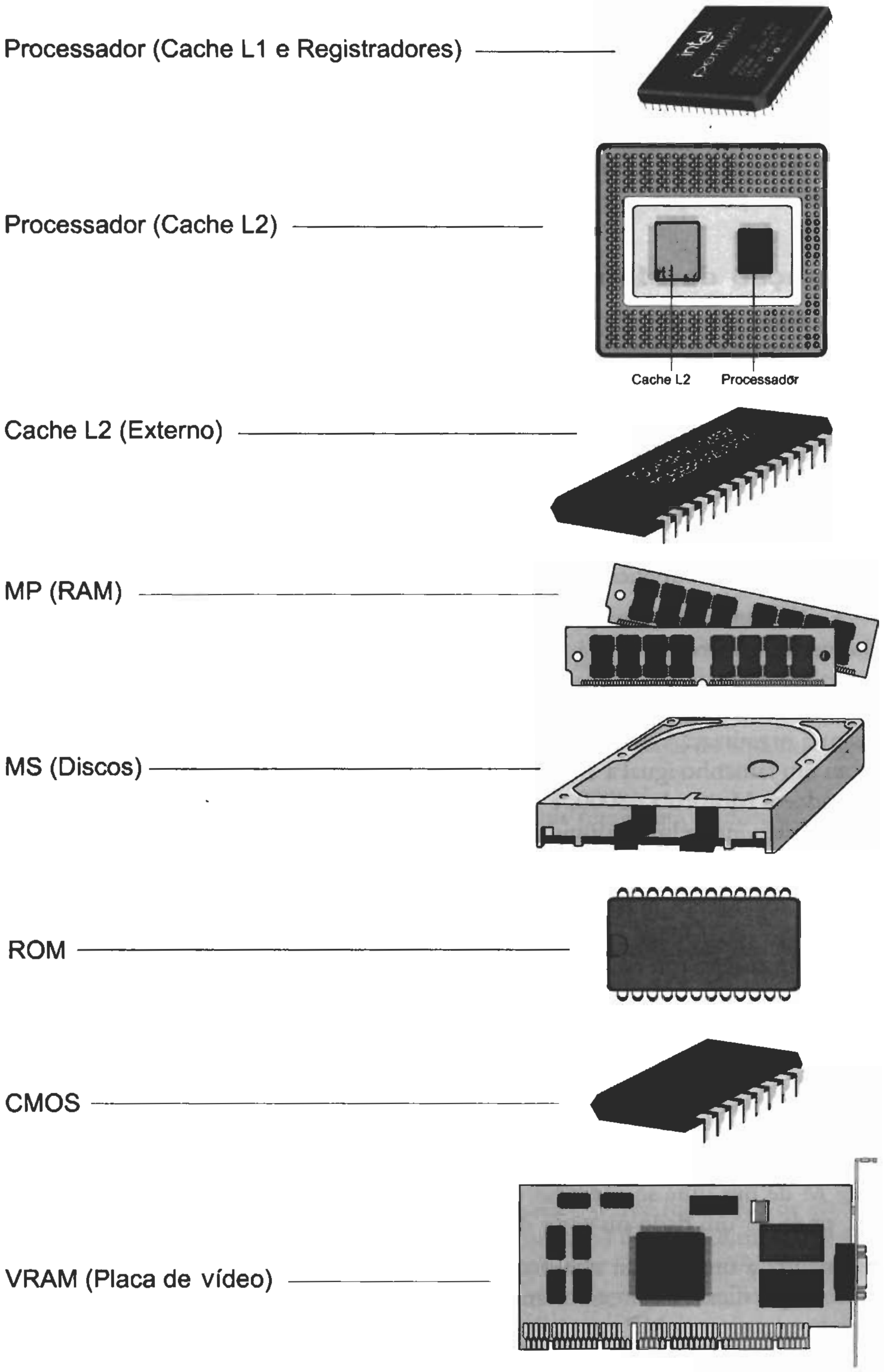

Processador (Cache L1 e Registradores)
intel pentium
Processador (Cache L2)
Cache L2
Processador
Cache L2 (Externo)
MP (RAM)
MS (Discos)
ROM
CMOS
VRAM (Placa de vídeo)

Figura 5.6 Tipos de memória em uso nos microcomputadores.

A Fig. 5.6 apresenta diversos tipos de memória atualmente em uso nos microcomputadores, caracterizando definitivamente a hierarquia e os elementos mencionados até então neste texto.

5.3 MEMÓRIA PRINCIPAL — MP

Neste item serão apresentadas as características essenciais da memória principal ou primária de um sistema de computação. Em primeiro lugar será mostrada a estrutura organizacional da MP, projetada para um acesso rápido e fácil pela UCP. Em seguida, serão descritas as operações que podem ser realizadas entre UCP/MP, bem como detalhes de capacidade, tempo de acesso e ciclo de máquina. Como conclusão, serão apresentados alguns tipos de memória principal.

5.3.1 Organização da Memória Principal

Para melhor descrever a organização da memória principal — MP dos computadores, é bom lembrar alguns conceitos já expostos juntamente com outros a serem apresentados pela primeira vez:

a) a MP é o "depósito" de trabalho da UCP, isto é, a UCP e a MP trabalham íntima e diretamente na execução de um programa. As instruções e os dados do programa ficam armazenados na MP e a UCP vai "buscando-os" um a um à medida que a execução vai se desenrolando;

b) os programas são organizados de modo que os comandos são descritos seqüencialmente e o armazenamento das instruções se faz da mesma maneira, fisicamente seqüencial (embora a execução nem sempre se mantenha de forma seqüencial — ver Cap. 6);

c) *palavra* — é a unidade de informação do sistema UCP/MP que deve representar o valor de um número (um dado) ou uma instrução de máquina. Desse modo, a MP deveria ser organizada como um conjunto seqüencial de palavras, cada uma diretamente acessível pela UCP. Na prática isso não acontece porque os fabricantes seguem idéias próprias, não havendo um padrão para o tamanho da palavra e sua relação com a organização da MP. Por exemplo, os antigos processadores Intel 8086/8088 possuíam palavra com um tamanho igual a 16 bits; a palavra dos processadores Intel 80486 e Pentium, bem como dos processadores Motorola 68000, é igual a 32 bits, enquanto a MP associada a todos esses processadores é organizada em células (a unidade de armazenamento) com 8 bits de tamanho.

d) *endereço, conteúdo e posição de MP* — em toda organização composta de vários elementos, que podem ser identificados e localizados individualmente para, com eles, ser realizado algum tipo de atividade, há necessidade de se estabelecer um tipo qualquer de identificação para cada elemento e associar a esta identificação um código (ou coisa parecida) que defina sua localização dentro da organização, de modo que cada elemento possa ser facilmente identificado e localizado. Este é o conceito, já exposto, de endereço, ou posição de MP (vale, aliás, para qualquer tipo de memória) (ver item 5.1). A Fig. 5.7 mostra um exemplo dos significados de endereço e conteúdo de uma memória e sua óbvia diferença.

e) *unidade de armazenamento* — consiste no grupo de bits que é inequivocamente identificado e localizado por um endereço. A MP é, então, organizada em unidades de armazenamento, denominadas *células,* cada uma possuindo um número de identificação — seu *endereço* — *e* contendo em seu interior uma quantidade M de bits, que se constitui na informação propriamente dita (pode ser uma instrução ou parte dela, pode ser um dado ou parte dele).

Teoricamente, a unidade de armazenamento da MP deveria ser a palavra, isto é, palavra e célula deveriam ser especificadas com o mesmo tamanho. Isto, na prática, não acontece, pois os fabricantes têm preferido organizar as MP com células de 1 byte (8 bits) de tamanho, com palavras de 16, 32 e até 64 bits. Há autores que definem célula como o local destinado a armazenar 1 bit e, nesse caso, um grupo de células (ou de bits) é acessado por um único endereço.

f) *unidade de transferência* — de ou para a MP, consiste na quantidade de bits que é transferida da memória em uma operação de leitura ou transferida para a memória em uma operação de escrita. Também, teoricamente, deveria ser igual à palavra e à unidade de armazenamento, porém na prática é possível encontrar computadores cuja integração UCP/MP é realizada com uma unidade de transferência diferente não só da palavra, mas também do tamanho da célula (unidade de armazenamento), seja por razões técnicas, seja apenas por interesses comerciais.

O processador Intel Pentium original, por exemplo, possuía palavra de 32 bits, barramento de dados de 32 bits (e, portanto, transferia 32 bits em cada operação de escrita ou leitura) e cada célula de memória armazenava 8 bits de informação[5] (ver Tabela 6.1).

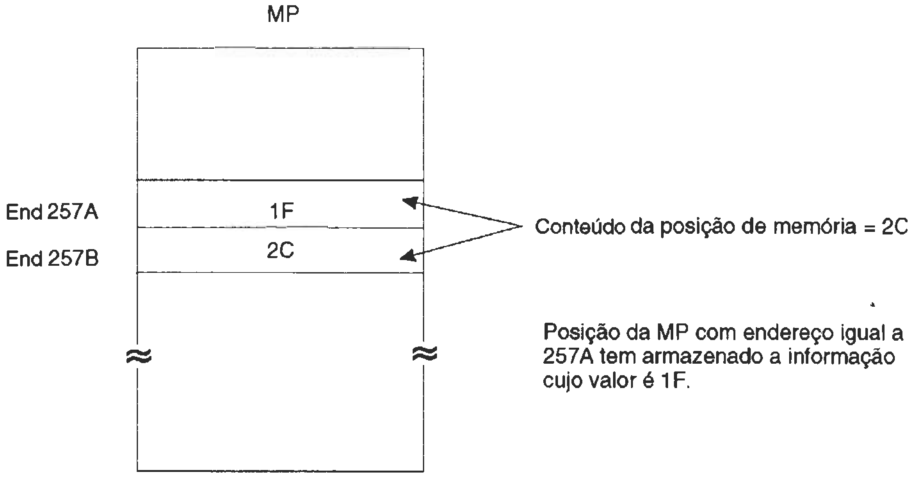

MP
End 257A
End 257B
1F
2C
Conteúdo da posição de memória = 2C
Posição da MP com endereço igual a 257A tem armazenado a informação cujo valor é 1F.

Figura 5.7 Significado dos valores de endereço e conteúdo na MP.

A memória principal de qualquer sistema de computação é organizada como um conjunto de N células seqüencialmente dispostas a partir da célula de endereço igual a 0 até a última, de endereço igual a N–1, conforme mostrado na Fig. 5.8.

Cada célula é construída para armazenar um grupo de M bits, que representa a informação propriamente dita e que é manipulado em conjunto (como se fosse uma única unidade) em uma operação de leitura ou de escrita.

Com o desenvolvimento da tecnologia de microeletrônica e de semicondutores, as antigas memórias de núcleo magnético foram substituídas por dispositivos voláteis de estado sólido. As memórias de semicondutores possuem várias características interessantes que as **tornam extremamente vantajosas para constituírem-se na** base da MP, quais sejam:

- são memórias de acesso aleatório (RAM — Random Access Memory);
- ocupam relativamente pouco espaço, podendo muitos bits ser armazenados em uma pastilha (*chip*); e
- possuem tempo de acesso pequeno.

Essencialmente, o espaço de armazenamento da memória principal (genericamente chamada de RAM) é um grupo de N células, cada uma podendo armazenar um grupo de M bits. Esta é a memória de trabalho da UCP e, portanto, deve permitir o armazenamento de instruções e dados (operação de escrita) e também a leitura destas mesmas instruções e dados. Chama-se a isso uma memória do tipo Leitura e Escrita (*Read/Write*). Este tipo de memória tem uma particularidade desvantajosa, que é o fato de ser volátil, isto é, perde toda a informação nela armazenada se for interrompida a energia elétrica que a alimenta.

No entanto, todo sistema precisa, para iniciar seu funcionamento regular, que um grupo de instruções (normalmente programas pequenos) esteja permanentemente armazenado na MP de modo que, ao ligarmos o computador, este programa inicie automaticamente o funcionamento do sistema. Essas instruções vêm junto com o hardware e não devem sofrer um acidental apagamento se, inadvertidamente, um programa do usuário tentar gravar em cima delas. Elas devem estar, portanto, em um tipo de RAM que só permita leitura por parte da UCP ou de outros programas. A gravação (escrita) nelas deve ser realizada eventualmente e não através de processos comuns. Essas memórias chamam-se *memórias somente de leitura* (ROM — Read Only Memory). O item 5.3.5 descreve outros aspectos relativos às memórias RAM e ROM.

---

[5] Atualmente, os processadores Pentium III utilizam o barramento externo de dados com um tamanho maior que o da palavra (é um múltiplo do valor de bits da palavra), de modo a otimizar o tempo de transferência de dados entre UCP e MP.

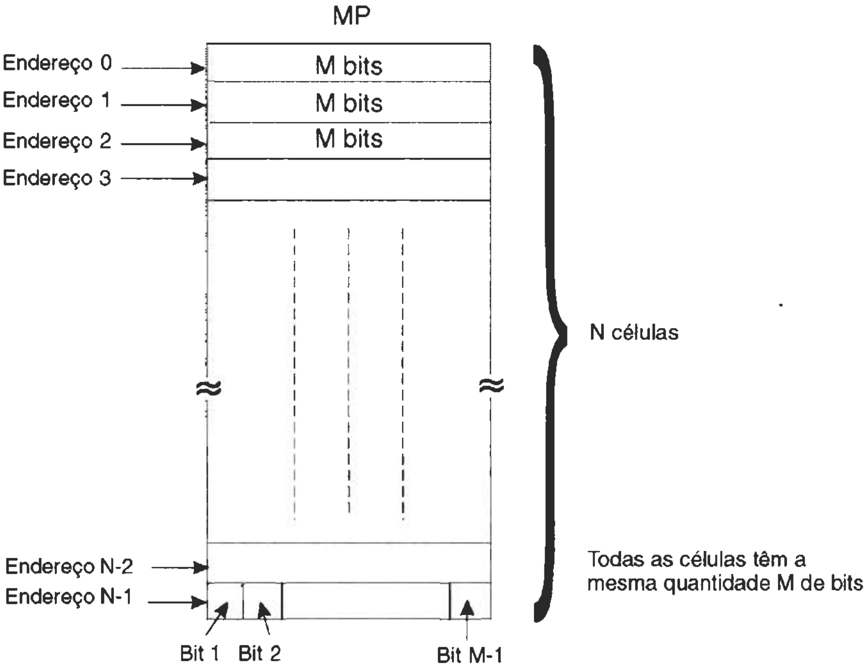


Figura 5.8 Organização básica da MP.

5.3.2 Considerações sobre a Organização da Memória Principal

Embora a organização estrutural das memórias de semicondutores, atualmente adotadas como MP, seja simples do ponto de vista conceitual — grupo de N células com M bits cada —, uma análise mais detalhada dessa mesma organização permite relacionar algumas observações interessantes que têm servido de base para modelos diferentes ou discussões técnicas.

a) *A quantidade de bits de uma célula* (valor de M na Fig. 5.8)

Já mencionamos que cada célula é constituída de um conjunto de circuitos eletrônicos, baseados em semicondutores, que permitem o armazenamento de valores 0 ou 1, os quais representam um dado ou uma instrução.

A quantidade de bits que pode ser armazenada em cada célula é um requisito definido pelo fabricante. Uma célula contendo M bits permite o armazenamento de 2^M combinações de valores, uma de cada vez, é claro. A Fig. 5.9 mostra alguns exemplos de MP com diferentes tamanhos de células.

Atualmente, praticamente todos os fabricantes vêm adotando um tamanho padrão de célula de 8 bits. No passado, no entanto, vários tamanhos foram utilizados. Na Tabela 6.1 estão relacionadas diversas características de memórias dos principais sistemas de computação.

b) *A relação endereço × conteúdo de uma célula*

A Fig. 5.9 mostra exemplos de memória com diferentes tamanhos de célula, porém com a mesma quantidade de células (nos exemplos há três organizações de MP, cada uma com 256 células), endereçadas de 0_{10} a 255_{10} ou de 00_{16} a FF_{16} ou ainda de 00000000_2 a 11111111_2.

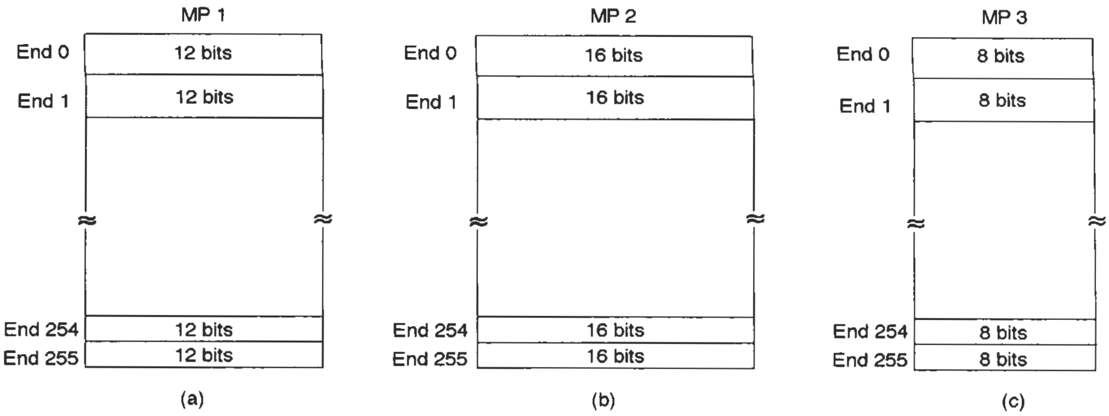

MP 1
End 0
12 bits
End 1
12 bits
End 254
12 bits
End 255
12 bits
(a)
MP 2
End 0
16 bits
End 1
16 bits
End 254
16 bits
End 255
16 bits
(b)
MP 3
End 0
8 bits
End 1
8 bits
End 254
8 bits
End 255
8 bits
(c)

Figura 5.9 Exemplos de MP com mesma quantidade de células (256), porém com largura de célula diferente.

A Fig. 5.10 mostra outros exemplos de organização de MP, dessa vez com memórias de mesmo tamanho de célula, porém com quantidades diferentes de células.

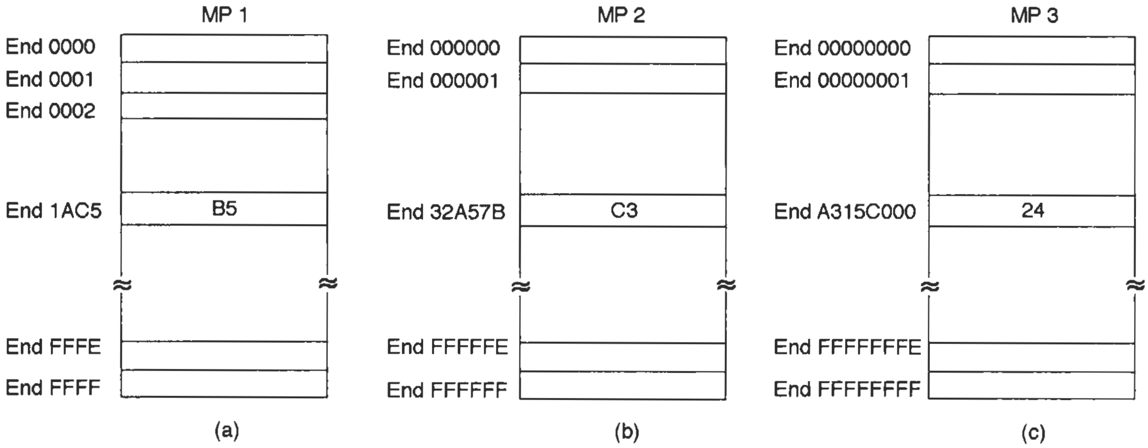

MP 1
End 0000
End 0001
End 0002
End 1AC5
B5
End FFFE
End FFFF
(a)
MP 2
End 000000
End 000001
End 32A57B
C3
End FFFFFE
End FFFFFF
(b)
MP 3
End 00000000
End 00000001
End A315C000
24
End FFFFFFFE
End FFFFFFFF
(c)

Figura 5.10 Exemplo de MP com mesma largura de célula, porém com quantidade de células diferente.

A comparação entre os exemplos das Figs. 5.9 e 5.10 indica que os valores de endereço e conteúdo de célula embora associados, ou seja, o endereço 1AC5 está associado ao conteúdo B5 no exemplo da Fig. 5.10(a), têm origens diversas. Em outras palavras, a quantidade de bits do número que representa um determinado endereço, por exemplo, 16 bits do número 1AC5 no exemplo da Fig. 5.10(a), define a quantidade máxima de endereços que uma MP pode ter, bem como o seu espaço de endereçamento. No exemplo citado, este espaço de endereçamento ou capacidade máxima da memória é 64K células, porque $2^{16} = 2^{6} \cdot 2^{10} = 64 \cdot \text{K}$ ou 64K.

Como todas as células têm o tamanho de 1 byte, a quantidade de células é sempre igual à quantidade de bytes.

5.3.3 Operações com a Memória Principal

Já sabemos, conforme anteriormente explicado, que é possível realizar duas operações em uma memória:

- escrita (*write*) — armazenar informações na memória;
- leitura (*read*) — recuperar uma informação armazenada na memória.

Sabemos também que a operação de leitura não destrói o conteúdo da memória, ela apenas providencia a transferência de uma cópia do que está armazenado, enquanto a informação desejada continua armazenada. Somente a operação de escrita é destrutiva.

Vamos descrever, com um pouco mais de detalhe, como se desenrola uma operação de leitura e uma de escrita na MP de um sistema de computação.

Para tanto, há necessidade de se definirem os elementos que compõem a estrutura UCP/MP e que são utilizados naquelas operações (ver Fig. 5.11):

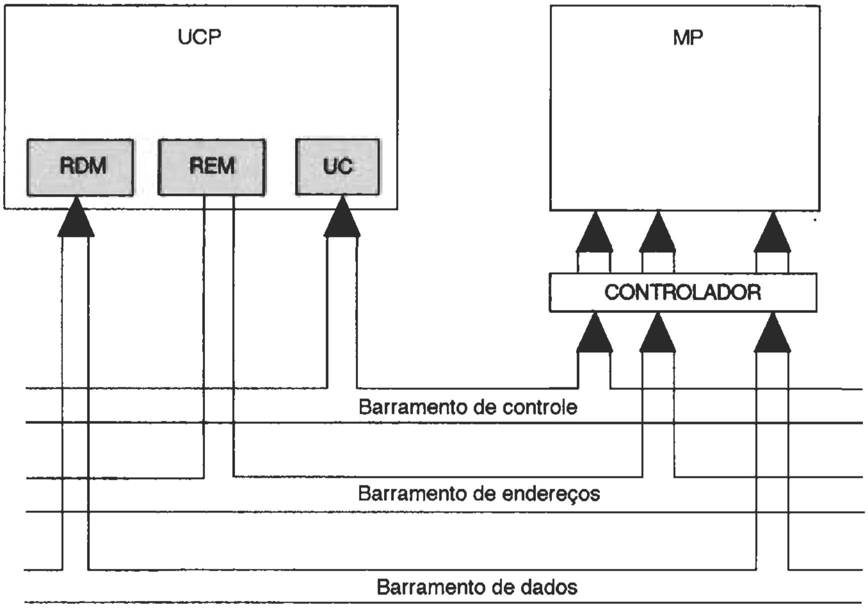


Figura 5.11 Estrutura UCP/MP e a utilização de barramento para comunicação entre eles.

Barramento de dados (ver item 6.6.3) — interliga o RDM (MBR) à MP, para transferência de informações entre MP e UCP (sejam instruções ou dados). É *bidirecional,* isto é, ora os sinais percorrem o barramento da UCP para a MP (operação de escrita), ora percorrem o caminho inverso (operação de leitura).

Registrador de Dados da Memória (*RDM*) (Memory Buffer Register — MBR) — registrador que armazena temporariamente a informação (conteúdo de uma ou mais células) que está sendo transferida da MP para a UCP (em uma operação de leitura) ou da UCP para a MP (em uma operação de escrita). Em seguida, a referida informação é reencaminhada para outro elemento da UCP para processamento ou para uma célula da MP, conforme o tipo da operação de transferência. Permite armazenar a mesma quantidade de bits do barramento de dados.

Registrador de Endereços da Memória (*REM*) (Memory Address Register — MAR) — registrador que armazena temporariamente o endereço de acesso a uma posição de memória, ao se iniciar uma operação de leitura ou de escrita. Em seguida, o referido endereço é encaminhado à área de controle da MP para decodificação e localização da célula desejada. Permite armazenar a mesma quantidade de bits do barramento de endereços.

Barramento de endereços — interliga o REM (MAR) à MP para transferência dos bits que representam um determinado endereço. É unidirecional, visto que somente a UCP aciona a MP para a realização de operações de leitura ou escrita. Possui tantos fios (ou linhas de transmissão) quantos são os bits que representam o valor de um endereço.

Barramento de controle — interliga a UCP (unidade de controle) à MP para passagem de sinais de controle durante uma operação de leitura ou escrita. É bidirecional, porque a UCP pode enviar sinais de controle para a MP, como sinal indicador de que a operação é de leitura (READ) ou de escrita (WRITE), e a MP pode enviar sinais do tipo WAIT (para a UCP aguardar o término de uma operação).

Controlador — também conhecido como Decodificador. Tem por função gerar os sinais necessários para controlar o processo de leitura ou de escrita, além de interligar a memória aos demais componentes do sistema de computação. É o controlador que decodifica o endereço colocado no barramento de endereços, localizando a célula desejada.

Para simplificar a descrição dos procedimentos de leitura/escrita, iremos omitir o papel do controlador naqueles procedimentos, deixando para o item 5.6 — Um pouco mais de detalhes — a apresentação e descrição de suas atividades.

Para simplificar a descrição de procedimentos e operações realizadas internamente em um sistema de computação vamos adotar uma convenção genericamente conhecida como LTR — Linguagem de Transferência entre Registradores (Register Transfer Language — RTL). Princípios básicos da LTR:

a) Caracteres alfanuméricos significam abreviaturas de nomes de registradores ou posições de memória. Ex.: REM, MP.

b) Parênteses indicam conteúdo, no caso de registradores, ou que o valor entre parênteses é um endereço de MP.

c) Uma seta indica atribuição, isto é, transferência de conteúdo de um registrador para outro ou para a MP, ou vice-versa.

Por exemplo:

(REM) (CI) — significa que o conteúdo do registrador cujo nome é CI é copiado para o registrador REM.
(RDM) (MP(REM)) — significa que o conteúdo da célula da MP cujo endereço está no REM é copiado para o RDM.

5.3.3.1 Operação de Leitura (Fig. 5.12)

A realização de uma operação de leitura é efetivada através da execução de algumas operações menores (microoperações), cada uma consistindo em uma etapa ou passo individualmente bem definido. O tempo gasto para realização de todas estas etapas caracteriza o *tempo de acesso* (ver item 5.2). O intervalo de tempo decorrido entre duas operações consecutivas (leitura-leitura, leitura-escrita ou escrita-leitura) denomina-se *ciclo de memória.*

A Fig. 5.12 mostra um exemplo de operação de leitura de um dado armazenado no endereço 1324 da MP (o valor do dado é 5C para a UCP).

Os passos que descrevem a referida operação de leitura são:

1) REM ← de outro registrador da UCP (ver Cap. 6)

1a) O endereço é colocado no barramento de endereços

2) Sinal de leitura é colocado no barramento de controle

2a) Decodificação do endereço e localização da célula

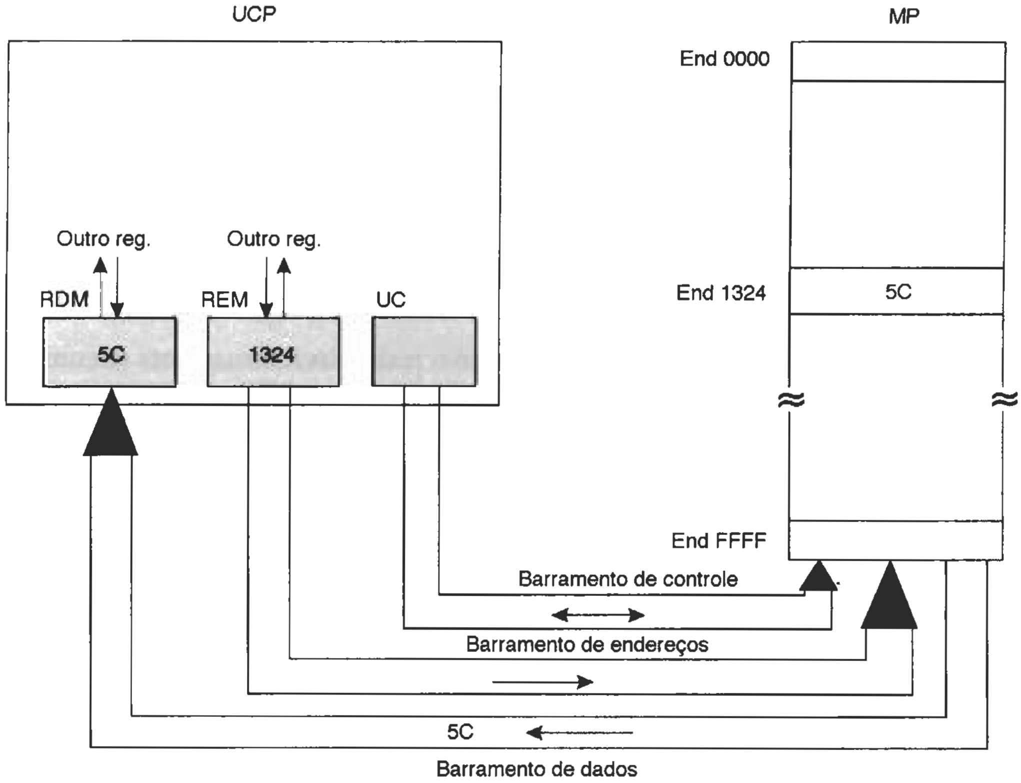
3) RDM ← MP(REM) pelo barramento de dados

4) Para outro registrador da UCP ← RDM

No primeiro passo, a Unidade de Controle — UC da UCP (a UC será descrita no Cap. 6) inicia a operação de leitura através da cópia do endereço 1324 de um de seus registradores específicos (pode ser, por exemplo, o CI, a ser descrito no Cap. 6) para o REM, e coloca o sinal de leitura (READ) no barramento de controle para indicar aos circuitos de controle da MP o que fazer em seguida.

A MP decodifica o endereço recebido (ver item 5.6.2) e copia seu conteúdo para o RDM através do barramento de dados. Do RDM, então, a informação desejada é copiada para o elemento da UCP, que é destinatário final (normalmente é um dos registradores da própria UCP).

A realização completa dos quatro passos descritos gasta um tempo de acesso, mas não garante que a MP possa realizar logo em seguida uma nova operação. Estar pronta ou não para realizar uma nova operação de-



1 - (REM) ← (outro reg.)
1a - O endereço é colocado no barramento de endereços
2 - Sinal de leitura no barramento de controle (decodificação)
3 - (RDM) ← (MP (REM))
4 - (outro reg.) ← (RDM)

Figura 5.12 Exemplo de operação de leitura.

pende do tipo de memória RAM utilizada, como veremos nos itens 5.3.5 e 5.6.1. As memórias estáticas (SRAM) permitem que outra operação (de leitura ou de escrita) seja imediatamente realizada após a conclusão de uma operação de leitura/escrita, enquanto memórias dinâmicas (DRAM), não.

5.3.3.2 Operação de Escrita (Fig. 5.13)

A realização de uma operação de escrita segue procedimento semelhante ao da operação de leitura, exceto, é claro, pelo sentido da cópia, que é inverso, isto é, da UCP para a MP.

A Fig. 5.13 mostra um exemplo de operação de escrita de um dado, de valor igual a F7, da UCP para a MP, a ser armazenado no endereço 21C8.

Os passos que descrevem a referida operação são:

1) (REM) ← (outro registrador) — a UCP coloca endereço no REM
1a) o endereço é colocado no barramento de endereços
2) (RDM) ← (outro registrador) — a UCP coloca no RDM o dado a ser copiado
3) Sinal de escrita é colocado no barramento de controle — a UCP aciona o sinal WRITE pelo barramento de controle
4) (MP(REM)) ← (RDM) — o dado é transferido para a célula de memória pelo barramento de dados.

Nos primeiros passos a UC coloca o endereço desejado no REM e o dado a ser copiado no RDM. O endereço é colocado no barramento de endereços, o dado no barramento de dados e o sinal de escrita (WRITE) é acionado no barramento de controle.

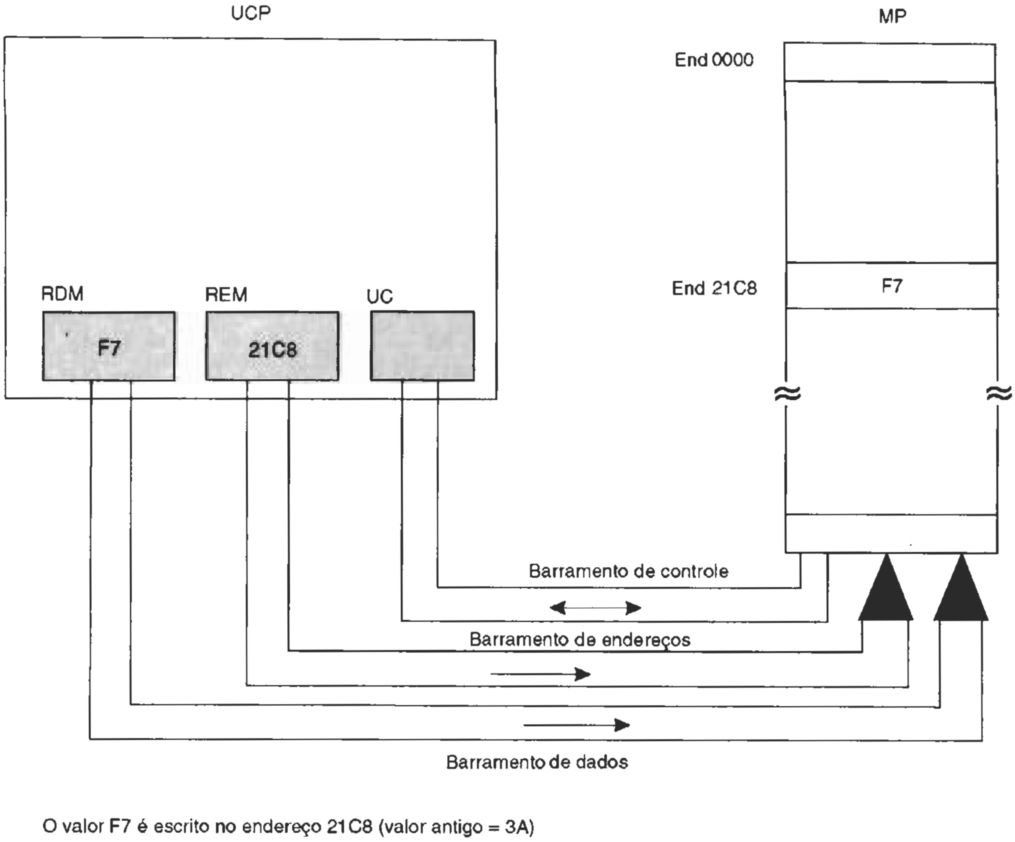

UCP
MP
End 0000
End 21C8
F7
RDM
REM
UC
F7
21C8
Barramento de controle
Barramento de endereços
Barramento de dados
O valor F7 é escrito no endereço 21C8 (valor antigo = 3A)

Figura 5.13 Exemplo de operação de escrita.

Como resultado da decodificação do endereço pelo dispositivo de controle da memória, o valor F7 é copiado na célula desejada, de endereço 21C8.

Conforme já foi explicado para a operação de leitura, a realização dos passos necessários à efetivação de uma operação de escrita gasta um tempo de acesso e a MP pode ou não estar preparada para realizar imediatamente nova operação.

Observação: A descrição dos passos relativos à realização de uma operação de leitura ou de escrita não teve o propósito de ser precisa no tempo e na sincronização absolutamente necessários se fosse desejada uma explicação mais profunda a respeito do mecanismo de sincronização de operações em um computador. No entanto, para quem estiver interessado neste nível de detalhe, o item 6.6.3 apresenta uma descrição mais precisa das referidas operações.

5.3.4 Capacidade de MP — Cálculos

Conforme já exposto neste capítulo, as memórias principais (MP), também comercialmente conhecidas como memórias RAM ou simplesmente RAM, são organizadas em conjuntos de células, cada uma podendo armazenar uma certa quantidade de bits — a informação em si, seja uma instrução (ou parte dela) ou um dado (ou parte dele).

Considerando que instruções e dados precisam estar armazenados na MP para que o programa possa ser executado pela UCP,[6] e considerando ainda que é possível, e bastante desejável, que vários programas possam ser executados em paralelo pela UCP, é importante conceituar o que seja capacidade de uma memória e como podemos calcular e entender aumentos de capacidade e outras informações concernentes. Neste item, o enfoque maior do assunto se refere aos cálculos de capacidade em memórias RAM (MP).

---

[6] Deve-se considerar que, atualmente, as técnicas utilizadas para gerenciamento de memória determinam que apenas uma parte do programa e seus dados precisam estar armazenados na MP durante a execução do referido programa. Esta parte tem diversas denominações, por exemplo, página.

Capacidade de memória refere-se genericamente à quantidade de informações que nela podem ser armazenadas em um instante de tempo. Tratando-se de um computador, cuja unidade básica de representação de informação é o bit, pode-se imaginar este elemento como unidade de medida de capacidade. Neste caso, poderia expressar a capacidade de uma memória com valores do tipo: 512 bits, 16.384 bits e 8.388.608 bits.

À medida que os valores crescem, torna-se mais complicado e pouco prático indicar o valor pela sua completa quantidade de algarismos. No item 2.2.1 foi mostrado que é possível simplificar esta informação através do emprego de unidades como o K (1K = 2^{10} = 1024), o M — mega (1M = 2^{20} = 1.048.576), o G — giga (1G = 2^{30} = 1.073.741.824), o T — tera (1T = 2^{40} = 1.099.511.627.776) e o P — Peta (1P = 2^{50} = 1024 T).

Desse modo, os mesmos valores antes indicados com todos os seus algarismos agora podem ser assim simplificados: 512 bits, 16K bits e 8M bits.

Mas, mesmo simplificando a apresentação da informação, continua-se, neste caso, a indicar a capacidade da memória pela quantidade de bits. No entanto, esta não é a melhor maneira de quantificar a referida capacidade.

Na realidade, sabemos que não é possível armazenar 2 (dois) ou mais valores em uma célula de memória, ou seja, em um único endereço somente um valor (um dado) poderá ser localizado e identificado. Isto porque se fossem armazenados dois valores em um endereço (uma célula), o sistema não saberia identificar qual dos dois seria o desejado em uma certa operação de leitura ou escrita (precisar-se-ia, então, de uma identificação a mais — um endereço dentro de um endereço), com todos os óbvios inconvenientes.

Dessa forma, o mais importante elemento para determinar a capacidade de uma memória é a quantidade de endereços que poderemos criar e manipular naquela memória, visto que, na melhor das hipóteses, pode-se armazenar um dado em cada endereço. Na realidade, o mais comum, principalmente quando se manipulam valores numéricos, é um dado ser armazenado ocupando várias células e, conseqüentemente, vários endereços. Por exemplo, se a MP de um certo sistema estiver organizada com células de 8 bits de tamanho e os dados forem definidos com 32 bits, então um dado será armazenado em 4 células (a maneira pela qual o sistema recebe a informação de endereço daquele dado não é discutida aqui).

Como não se pode armazenar dois números no mesmo endereço (mas um número pode até ocupar mais de um endereço), a quantidade de endereços tem mais sentido de individualidade de informação do que qualquer outra unidade.

Na prática, então, usa-se a quantidade de células para representar a capacidade da memória e, como no mercado informal de compra, venda, assistência técnica etc. da maioria dos computadores a célula de memória principal (que o mercado denomina informalmente RAM) tem um tamanho de 8 bits — 1 byte —, usa-se mesmo é a quantidade de bytes (e muitos abreviam a informação colocando apenas o valor, sem a unidade, de tão comum que é o uso de bytes). Em outras palavras, é normal procurar-se memória para compra informando ao vendedor: "preciso de 16 megas", o que significa, na realidade: preciso de memória com 16 megacélulas de 1 byte cada uma, isto é, preciso de 16 megabytes de memória.

Ainda mais:

a) o bit, apesar de ser a unidade elementar de representação de informação nos computadores, individualmente não representa nenhuma informação útil (com exceção do valor de uma variável lógica), pois somente pode assumir dois valores, 0 ou 1;

b) para representar uma informação útil, seriam necessários mais bits (em grupo) para se poder codificar vários elementos de um conjunto qualquer. Por exemplo, a representação de caracteres maiúsculos, minúsculos, séries de pontuação, gráficos etc. requer, em média, 7 ou 8 bits, de modo a se criar um conjunto de 128 ou 256 códigos, conforme já mencionamos no início deste capítulo;

c) os tempos de transferência de informações entre UCP e MP (tempo de acesso) são decorrentes de vários fatores (tipo de circuitos para construção da memória, distância física entre os elementos, quantidade de etapas durante as operações de transferência, duração do pulso do relógio da UCP etc. — ver Cap. 6 e item 5.6), dos quais o menos importante é a quantidade de bits. Deve ser esclarecido que estamos falando de um único acesso e não de vários acessos em conjunto, pois, neste último caso, o desempenho do sistema aumenta sensivelmente se o barramento de dados for maior.

Uma outra possível unidade de medida de capacidade de memória seria a *palavra*. Expressar uma capacidade da forma 16K palavras é mais significativo do que 16 Kbytes visto que, neste último caso, não sabemos ao certo quantas palavras há na tal memória (quantos dados ou instruções podem ser armazenados na memória), enquanto, no primeiro exemplo, a palavra deve indicar o tamanho de um dado ou de uma instrução (na prática, isto também não é verdade, o que complica muitas vezes os cálculos de capacidade).

Não há uma padronização para indicar valores de capacidade de memória, embora seja mais comum se usar "quantidade de bytes" em vez de "quantidade de palavras". Para computadores construídos com propósitos comerciais usa-se o byte como unidade básica de armazenamento (embora isto não seja um padrão imutável), enquanto computadores ditos científicos, mais habilitados a manipular números (como os supercomputadores), costumam organizar sua MP em palavras.

No entanto, a título de esclarecimento e para acostumar o leitor com os jargões da área, vamos apresentar a seguir alguns exemplos de expressões para representar capacidade de memória.

2 Kbytes = 2×2^{10} = 2048 bytes

384K células = 384×2^{10} = 393.216 células

384K palavras = 393.216 palavras

2 Mbytes = 2×2^{20} = 2.097.152 bytes.

5.3.4.1 Cálculos com Capacidade da MP (RAM)

A memória principal (RAM) é um conjunto de N células, cada uma armazenando um valor com M bits. Então, a quantidade de endereços contida no espaço endereçável da referida RAM é também igual a N, visto que a cada conteúdo de célula está associado um número, que é o seu endereço.

O valor de N representa a *capacidade da memória*, através da quantidade de células ou de endereços. O valor de M indica a quantidade de bits que pode ser armazenada em uma célula individual (que é a informação propriamente dita). Como 1 bit representa apenas um entre dois valores (base binária), então podemos concluir que:

a) pode-se armazenar em cada célula um valor entre 0 e 2^{M-1}, porém um de cada vez. São 2^M combinações possíveis.

Por exemplo, se M = 8 bits, temos $2^8 = 256$.

Seriam armazenados valores entre:

00000000(0_{10} ou 0_{16}) e 11111111 (255_{10} ou FF_{16}).

b) a MP tendo N endereços e sendo E = quantidade de bits dos números que representam cada um dos N endereços, então:

$N = 2^E$.

Por exemplo, se N = 512 (porque a MP tem 512 células), então, $512 = 2^E$, e E = 9, pois $2^9 = 512$.

c) o total de bits que podem ser armazenados na referida MP é denominado T, sendo:

$T = N \times M = 2^E \times M$

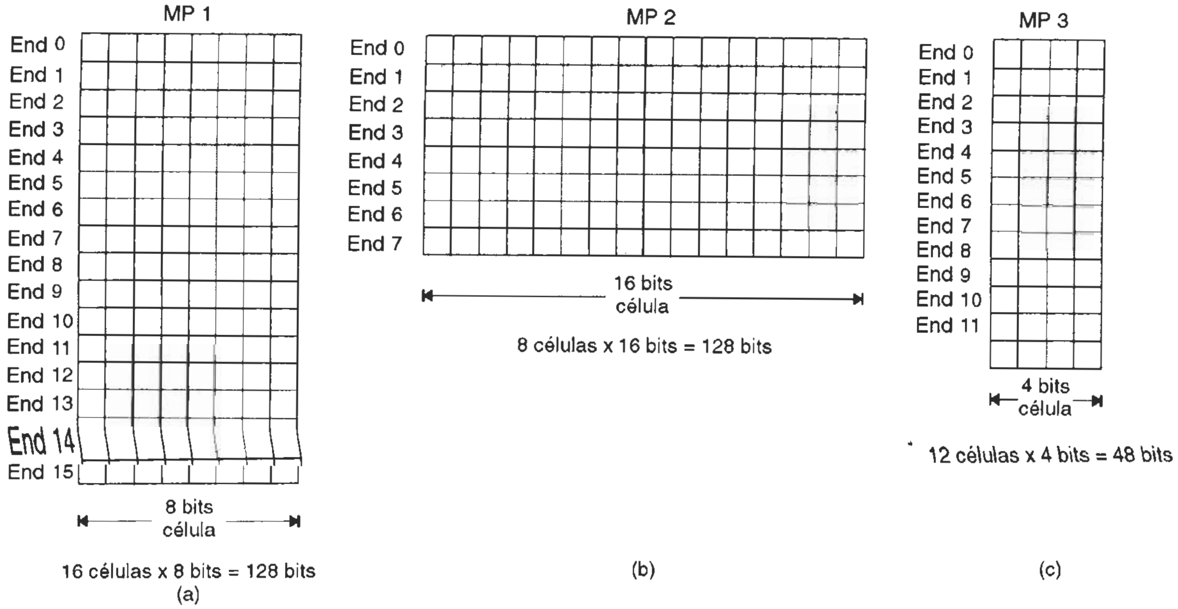
No exemplo em que a MP (RAM) é um espaço seqüencial de 512 células, cada uma com 8 bits de tamanho, teremos:

N (total de células) = 512 células; M (tamanho de cada célula) = 8 bits; E (tamanho em bits do número que representa cada endereço) = 9 bits; T (total de bits da memória) = 4096 bits.
$N = 2^E$; $512 = 2^E$ donde E = 9.
$T = N \times M = 512 \times 8 = 4096$ bits = 4×1024 (K) bits ou 4K bits.

Também poder-se-ia obter o valor 4K utilizando potenciação:

$512 \times 8 = 2^9 \times 2^3 = 2^{12} = 2^2 \times 2^{10} = 4$ K, pois $2^2 = 4$ e 2^{10} = K.

A Fig. 5.14 mostra exemplos de configurações de MP (RAM) com diferentes valores de N, M, E e T.


MP 1
End 0
End 1
End 2
End 3
End 4
End 5
End 6
End 7
End 8
End 9
End 10
End 11
End 12
End 13
End 14
End 15
8 bits
célula
16 células x 8 bits = 128 bits
MP 2
End 0
End 1
End 2
End 3
End 4
End 5
End 6
End 7
16 bits
célula
8 células x 16 bits = 128 bits
MP 3
End 0
End 1
End 2
End 3
End 4
End 5
End 6
End 7
End 8
End 9
End 10
End 11
4 bits
célula
12 células x 4 bits = 48 bits

(a) (b) (c)

Figura 5.14 Exemplos de organização de MP.

Exemplo 5.1

Uma memória RAM (MP) tem um espaço máximo de endereçamento de 2K. Cada célula pode armazenar 16 bits. Qual o valor total de bits que pode ser armazenado nesta memória e qual o tamanho de cada endereço?

Solução

Se o espaço máximo endereçável é 2K, então: N = 2K (a quantidade máxima de células é 2K).

1 célula = 16 bits. Então: M = 16 bits (tamanho em bits de cada célula).

Sendo $N = 2^E$, então: $N = 2K = 2 \times 1024$ e convertendo em potências de 2, temos: $2^1 \times 2^{10} = 2^{11}$.

Se $N = 2^E$ e $N = 2^{11}$, então: $2^E = 2^{11}$ e $E = 11$.

Se E = quantidade de bits de cada número que expressa um endereço, e sendo E = 11, então os endereços de cada célula são números que têm 11 bits.

$T = N \times M = 2^{11} \times 16 = 2^{11} \times 2^4 = 2^{15}$. Convertendo para múltiplo de $K = 2^{10}$, teremos: $2^5 \times 2^{10} = 32K$.

Respostas: Total de bits da MP: 32K (T)

Tamanho de cada endereço: 11 bits (E)

Exemplo 5.2

Uma memória RAM (MP) é fabricada com a possibilidade de armazenar um máximo de 256K bits. Cada célula pode armazenar 8 bits. Qual é o tamanho de cada endereço e qual é o total de células que podem ser utilizadas naquela RAM?

Solução

Total de bits = T = 256K. Utilizando potenciação, temos: $256K = 2^8 \times 2^{10} = 2^{18}$.

1 célula = 8 bits. Então: M (tamanho de cada célula) = 8 = 2^3.

Sendo T = N × M, então: N (quantidade de células) = T/M = 256K / 8 = 32K.

Pode-se obter o mesmo resultado através de potenciação: $256K/8 = 2^{18}/2^3 = 2^{15} = 2^5 \times 2^{10} = 32K$.

Se $N = 2^{15}$ e se sabemos que $N = 2^E$, então: E = 15.

Respostas: Tamanho de cada endereço: 15 bits (E)
Total de células: 32K (N)

Nesse instante podemos passar a considerar também os elementos básicos do processo de transferência de dados entre MP e UCP: REM, RDM, barramento de dados e barramento de endereços. Além disso, devem ser efetuadas algumas considerações sobre os tamanhos de cada um desses elementos, o que permitirá realizar os cálculos de capacidade desejados.

Como o BD — barramento de dados — interliga o RDM e a memória RAM (MP), então ambos possuem o mesmo tamanho em bits. O mesmo acontece entre o BE — barramento de endereços — e o REM.

Como o REM e o BE — barramento de endereços — têm por função armazenar o endereço de acesso a uma célula da MP, então seus tamanhos devem corresponder à quantidade de bits de cada endereço. Em outras palavras, o tamanho, em bits, do REM e do BE é igual ao valor de E na equação $N = 2^E$.

Já o RDM deve ter um tamanho correspondente à palavra do sistema, visto que deve transferir os bits de uma palavra entre a UCP e a MP (ou vice-versa, dependendo de a operação ser de escrita ou de leitura). Na prática, não há um padrão de tamanho de RDM (ou seja, nem sempre seu tamanho é o da palavra) e, portanto, para cada caso deve ser indicado o seu tamanho ou uma indicação de como obtê-lo. A título de informação pode-se mencionar que, atualmente, a maioria dos processadores possui um BD — barramento de dados — com tamanho múltiplo da palavra, de modo a acelerar o processo de transferência de dados entre UCP/MP, visto que a UCP tem espaço de armazenamento interno para receber dados antes que esses sejam processados.

Exemplo 5.3

Um computador, cuja memória RAM (MP) tem uma capacidade máxima de armazenamento de 2K palavras de 16 bits cada, possui um REM e um RDM. Qual o tamanho destes registradores? Qual o valor do maior endereço dessa MP e qual a quantidade total de bits que nela pode ser armazenada?

Solução

Se a capacidade máxima da MP é 2K palavras, entende-se que o endereçamento é por palavra (nada foi dito diferente) e, então: N = 2K.

Se cada palavra tem 16 bits e foi entendido que o endereçamento é por palavra, deduz-se que em cada célula pode ser armazenada uma palavra. Nesse caso, M = 16.

Sabe-se que $N = 2^E$ e, no exemplo, N = 2K. Então: $2^E = 2K = 2^1 \times 2^{10} = 2^{11}$. E = 11 bits.

Se cada endereço é um número de 11 bits, então o REM (registrador cuja função é armazenar endereços) também deve ter um tamanho igual a 11 bits.

Se a palavra tem 16 bits e o RDM é o registrador cuja função é armazenar uma palavra de dados (não foi especificado outro dado), então RDM = 16 bits.

Como T = N × M, então: T = 2K × 16 = 32K bits.

Finalmente, o maior endereço deve ser igual a (2K − 1) ou (2 × 1024) − 1 ou 2048 − 1 = 2047.

Respostas: Tamanho do REM = 11 bits.
Tamanho do RDM = 16 bits.
Maior endereço: $11111111111_2 = 2047_{10} = 7FF_{16}$.
Quantidade total de bits da MP: 32K bits.

Exemplo 5.4

Um processador possui um RDM com capacidade de armazenar 32 bits e um REM com capacidade de armazenar 24 bits. Sabendo-se que em cada acesso são lidas 2 células da memória RAM (MP) e que o barramento de dados (BD) tem tamanho igual ao da palavra, pergunta-se:

a) Qual é a capacidade máxima de endereçamento do microcomputador em questão?

b) Qual é o total máximo de bits que podem ser armazenados na memória RAM (MP)?

c) Qual é o tamanho da palavra e de cada célula da máquina?

Solução

RDM = 32 bits e REM = 24 bits BD = palavra cada acesso = 2 células

A capacidade máxima de endereçamento, isto é, total de endereço e de células é igual a $N = 2^E$.

Se REM = 24 bits e se REM armazena valor de endereço, então: E = 24 bits e $2^E = 2^{24}$.

Separando as potências, teremos: $2^{24} = 2^4 \times 2^{20} = 16 \times 1M$ (pois $1M = 2^{20}$).

Assim, N = 16M endereços ou 16M células. (Resposta *a*)

Total de bits = T = N × M, sendo M = tamanho de 1 célula e N = total de células = 16M = 2^{24}.

Como em cada acesso se lêem 2 células, e um acesso transfere, pelo BD, uma quantidade de bits, então: BD = 2 células. Como BD = RDM, então: BD = 32 bits = 2 células. Uma célula = ½ BD = 16 bits.

Então: T = 16M células × 16 bits = 256M bits. Por potenciação temos: $2^{24} \times 2^4$ (pois $16 = 2^4$) = 2^{28}.

Como $2^{28} = 2^8 \times 2^{20}$, então: 256M bits. (Resposta *b*)

O tamanho da palavra é igual ao do barramento de dados (BD). Como BD = RDM, então: palavra = 32 bits.

O tamanho de cada célula é de 16 bits, pois já vimos que célula = ½ BD (em cada acesso são lidas 2 células).

Respostas:

a) Capacidade máxima de endereçamento do microcomputador: 16M células.

b) Total máximo de bits que podem ser armazenados na memória RAM: 256M bits.

c) Tamanho da palavra: 32 bits e tamanho de cada célula: 16 bits.

Exemplo 5.5

Um processador possui um BE (barramento de endereços) com capacidade de permitir a transferência de 33 bits de cada vez. Sabe-se que o BD (barramento de dados) permite a transferência de quatro palavras em cada acesso e que cada célula da memória RAM (MP) armazena um oitavo (1/8) de cada palavra. Considerando que a memória RAM (MP) pode armazenar um máximo de 64G bits, pergunta-se:

a) Qual é a quantidade máxima de células que podem ser armazenadas na memória RAM (MP)?

b) Qual é o tamanho do REM e BD existentes neste processador?

c) Qual é o tamanho de cada célula e da palavra desta máquina?

Solução

BE = 33 bits   BD = 4 palavras por acesso   célula = 1/8 da palavra   T = 64G bits

Sabemos, por definição, que BE = REM, BD = RDM, M = tamanho de cada célula e T = N × M.

Assim, REM = 33 bits. Como, então, cada um dos N endereços da MP é um número de 33 bits, E = 33, e

N sendo igual a 2^E, N = $2^{33} = 2^3 \times 2^{30}$ = 8G endereços ou células, pois $2^3 = 8$ e 2^{30} = 1G. (Resposta *a*)

Como T = N × M, e sendo M = tamanho de 1 célula, então: M = T/N ou M = 64G (T)/8G (N) = 8 bits.

As unidades estão corretas, pois T = bits em células e N = total de células. Então, M = bits em células/célula = bits.

Se 1 célula = 8 bits e cada célula = 1/8 palavra, então: tamanho de 1 palavra = 8 × 1 célula = 8 × 8 = 64 bits.

Se o BD permite transferência de quatro palavras de cada vez, então: BD = 4 × 1 palavra = 4 × 64 = 256 bits.

Respostas:

a) Quantidade máxima de células que podem ser armazenadas na memória: 8G células.

b) Tamanho do REM: 33 bits e tamanho do BD: 256 bits.

c) Tamanho de cada célula: 8 bits e tamanho da palavra: 64 bits.

5.3.5 Tipos e Nomenclatura de MP

A memória principal dos computadores modernos é fabricada com tecnologia de semicondutores, o que lhes permite elevada velocidade de acesso e transferência de bits, já que são circuitos apenas elétricos em funcionamento (não há partes mecânicas, como acontece nos discos e fitas magnéticas, bem como nos disquetes e CD-ROMs). A velocidade de propagação de um sinal elétrico é nominalmente a velocidade da luz (300.000 km/s).

Tais memórias, no entanto, mantêm os valores binários armazenados apenas enquanto estiverem energizadas, sendo assim do tipo volátil, conforme já foi mencionado anteriormente. O item 5.6 descreve em detalhe métodos de construção e funcionamento dessas memórias.

A memória principal é a memória de trabalho da UCP, seu grande "bloco de rascunho", onde os programas (e seus dados) se sucedem em execução, uns após os outros. Ou seja, para que um programa seja executado, é necessário que suas instruções e os dados por elas manipulados estejam armazenados, ainda que temporariamente, na memória principal (MP). Este programa e dados estão normalmente armazenados de forma permanente na memória secundária, seja um disco magnético (usualmente denominado disco rígido, do inglês *hard disk* — HD), seja um CD-ROM.

Por exemplo, quando vamos trabalhar um texto utilizando um processador de textos, tipo Microsoft Word, é necessário que o código do Word esteja armazenado na MP para garantir velocidade no processamento (a UCP precisa encontrar na MP as instruções do Word necessárias a um determinado processamento do texto (colocar negrito em uma palavra, por exemplo), assim como o texto trabalhado deve estar armazenado na MP.

Na realidade, as coisas ocorrem de modo ligeiramente diferente, visto que atualmente não é mais necessário que o programa inteiro esteja armazenado na MP. Basta que ele seja dividido em "pedaços", chamados páginas, e que o sistema transfira apenas algumas das páginas de cada vez da memória secundária para a MP (as que estão ou irão ser usadas em breve). Este método será discutido no capítulo que trata da memória secundária. Além disso, como já mencionamos antes, o processador de texto e o arquivo de dados (o texto) estão inicialmente armazenados em disco. E, mais ainda, atualmente a UCP não acessa diretamente a MP, como nos sistemas mais antigos: ela procura, inicialmente, a instrução desejada no momento ou o dado requerido para um processamento, em outro tipo de memória, a memória cache, já apresentada anteriormente e que será mais bem descrita no item 5.5.

A Fig. 5.15 mostra um demonstrativo do fluxo de bits para um determinado processamento, a partir de seu local de armazenamento permanente (o disco rígido) até sua chegada na UCP para se efetivar o processamento.

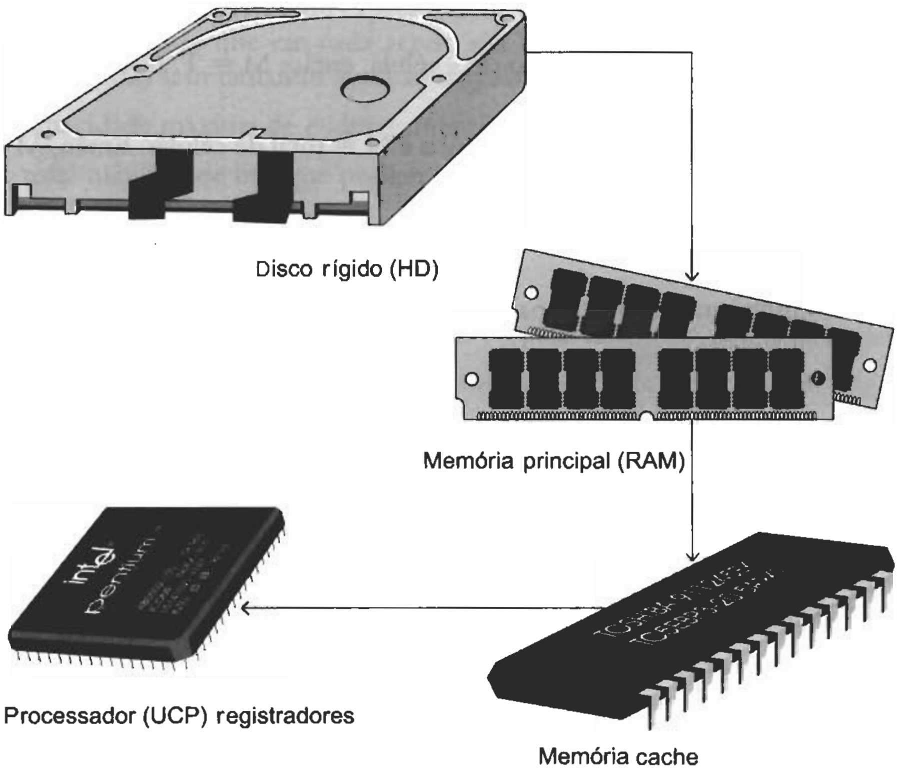


Figura 5.15 Fluxo de bits para um processamento.

A MP dos microcomputadores é comercial e popularmente denominada memória RAM ou simplesmente RAM. O termo é uma sigla das palavras inglesas Random Access Memory, cuja tradução é Memória de Acesso Aleatório (ou randômico).

As memórias RAM são construídas com tecnologia que lhes garante tempos de acesso na faixa dos nanossegundos (variável entre valores da ordem de 7 a 70 ns, de acordo com os elementos e processos de fabricação — ver Tabela 5.2, no item 5.6), tendo uma característica única e marcante: o tempo de acesso a qualquer de suas células é igual, independente da localização física da célula.

Em outras palavras, se o tempo de acesso de uma determinada memória de semicondutor é, por exemplo, 70 ns, isso significa que este será o tempo para acessar a célula de endereço 0 (1.ª célula). Se, em seguida, se desejar acessar a célula de maior endereço, o tempo também será igual a 70 ns, como também será de 70 ns se, após acessar a célula de endereço 13A, se desejar acessar a célula de endereço 13B. Qualquer que seja o endereço, aleatoriamente ou randomicamente escolhido, o tempo de acesso será o mesmo. Daí o seu nome — memória de acesso randômico (RAM).

Isso é possível devido à tecnologia (circuitos eletrônicos) de construção, da memória onde sinais elétricos percorrem os condutores (cujo comprimento é milimétrico) com a velocidade da luz, seja da primeira para a última célula, seja da terceira para a quarta. Não há movimento físico de qualquer de seus elementos (o que consumiria tempo), pois uma célula é localizada através da decodificação de seu endereço (processo quase instantâneo) e emissão de sinais elétricos correspondentes para "abertura" da célula e passagem dos seus bits para o barramento de dados.

A tecnologia RAM tem variações, que foram evoluindo com o tempo, as quais redundaram em vários diferentes tipos de memória. Estes tipos podem ser primeiramente grupados em duas vertentes: a SRAM (Static RAM) e DRAM (Dynamic RAM), isto é, a RAM estática e a RAM dinâmica. O primeiro tipo, mais rápido e de custo mais elevado, costuma ser utilizado na construção das memórias cache, e o outro tipo, DRAM, é aquele usado genericamente nas memórias principais tradicionais. Este tipo vem evoluindo e sendo produzido pelos diversos fabricantes com diferentes nuances, que lhe valeram diversos nomes, conforme mostrado na Tabela 5.2. Ambos os tipos, no entanto, são voláteis.

A memória RAM, constituída de memórias eletrônicas, de tempo de acesso igual independente da célula localizada, pode servir para construção de dois tipos de memória no que se refere à sua aplicação em um sistema de computação: memórias que servem para que seus conteúdos sejam lidos ou escritos (memórias L/E), denominadas em inglês *R/W memory*, e memórias em que os programas aplicativos podem apenas ler seu conteúdo, não lhes sendo permitido gravar em suas células, as memórias do tipo ROM (Read Only Memory ou memória somente para leitura). Estas têm uma notável particularidade que é o fato de não serem voláteis, como as memórias L/E.

Além de a MP permitir que um programa seja armazenado logo depois de outro (isto significa que são realizadas sucessivas operações de escrita nas mesmas células), durante a execução normal de um programa, suas instruções são sucessivamente lidas pela UCP, que, por sua vez, também realiza operações de escrita sobre a MP, armazenando resultados das operações realizadas.

Se uma memória é de acesso aleatório (RAM) para leitura, invariavelmente também o será para realizar ciclos de escrita. Assim, as memórias do tipo RAM, que permitem leitura/escrita (R/W), são usadas como memória principal (MP), e este termo (RAM) passou a ser tão comum que se confundiu com o próprio nome da memória (comumente se usa no comércio e na indústria o termo RAM quando se refere à MP, assim como falamos, em geral, gilete (de Gillette — nome do fabricante) por lâmina de barbear).

O uso do termo RAM para definir a memória principal de trabalho, onde nossos programas e aplicativos são armazenados, é incorreto, como veremos a seguir.

Embora seja rápida (tempo de acesso pequeno) e de acesso aleatório (mesmo tempo no acesso a qualquer célula), a RAM possui algumas desvantagens, entre as quais a *volatilidade,* isto é, perde seu conteúdo quando a corrente elétrica é interrompida.

Por outro lado, as memórias *read-write* apresentam o inconveniente (nem sempre é um inconveniente) de, permitindo que se escreva normalmente em suas células, ser possível a acidental eliminação do conteúdo de uma ou mais de suas células.

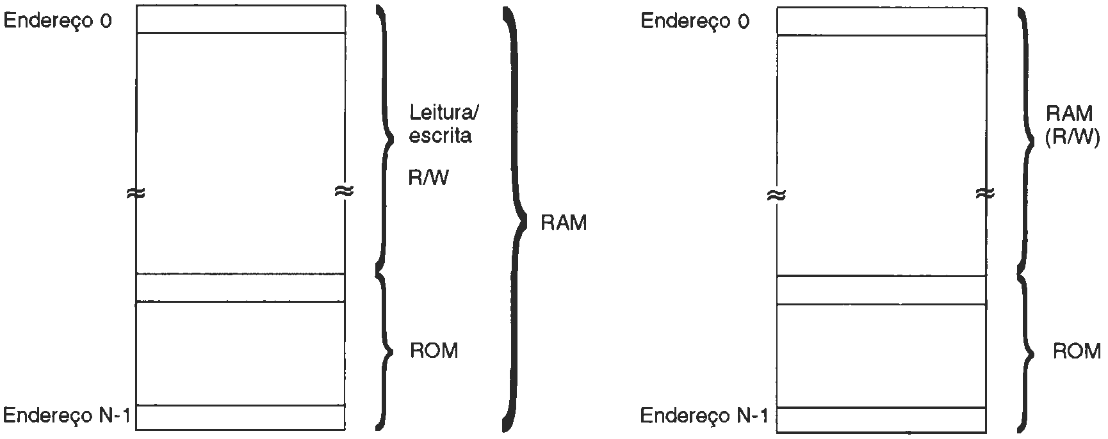
Uma vez que o processador nada executa sem que haja uma instrução especificando o que será executado, é óbvio que ele deve possuir uma certa quantidade de memória não-volátil, para armazenar as instruções iniciais. Isto é, um local onde estejam permanentemente armazenadas instruções que automaticamente iniciam a operação e a inicialização do sistema, tão logo a alimentação elétrica seja ligada.

Em microcomputadores costuma-se chamar de programa *bootstrap*, ou simplesmente *boot*, enquanto outros fabricantes chamam IPL — Initial Program Load (Carregamento do Programa Inicial), entre outros nomes.

Esse tipo de memória (ainda de semicondutores e, portanto, RAM), além de ter que ser *não-volátil* (para não haver a perda do programa de *boot*), também não deve permitir que haja eliminações acidentais. Trata-se de um programa que deve estar permanentemente armazenado e não sofrer alterações por parte de nenhum outro programa. Em outras palavras, memórias que armazenam este tipo de programas devem permitir *apenas leitura*. Chamam-se estas memórias de ROM — Read Only Memory (memórias somente para leitura) e elas devem ser não-voláteis.

No entanto, o tempo de acesso em memórias ROM também é constante, independentemente da localização física da célula e, por conseguinte, elas também são memórias RAM. Porém, o mercado incorreu no engano de chamar de RAM apenas as memórias R/W — leitura/escrita, talvez para claramente diferençá-las do outro tipo, ROM (somente para leitura), já que as siglas são bem parecidas.

A Fig. 5.16 apresenta a distribuição espacial das memórias R/W, RAM e ROM em um microcomputador, indicando o conceito correto e o conceito usado na prática pelo mercado.



(a) Nomenclatura correta

(b) Nomenclatura popular

Figura 5.16 Configuração da memória principal (MP) de um microcomputador do tipo PC.

As memórias DRAM vêm evoluindo em termos de aumento de capacidade e de velocidade (embora esta em menor percentagem do que o desejado em face do aumento da velocidade dos processadores), redundando em diversos tipos: EDO DRAM, BEDO DRAM, SDRAM, RDRAM e outros, conforme apresentado na Tabela 5.2 e no quadro-resumo a seguir.

Podemos classificar as memórias de semicondutores do seguinte modo:

- RAM
  - L/E — Leitura/Escrita (R/W — Read/ Write)
    - SRAM
    - DRAM
      - FPM DRAM
      - EDO DRAM
      - BEDO DRAM (ver Tabela 5.1)
      - SDRAM
      - RDRAM
  - ROM (Somente leitura) (Read Only Memory)
    - ROM
    - PROM
    - EPROM
    - EEPROM

Como podemos observar, tanto as memórias L/E (que permitem leitura e escrita) quanto as memórias ROM (que permitem apenas leitura) são memórias de acesso randômico, isto é, são memórias RAM.

A descrição dos diversos tipos de RAM e suas aplicações mais comuns encontram-se no item 5.6, enquanto os modelos de memórias ROM são descritos no item 5.3.5.1 a seguir.

5.3.5.1 Memórias do Tipo ROM

Memórias ROM são também memórias de semicondutores fabricadas para atingir três objetivos:

a) ter desempenho semelhante ao das memórias R/W de semicondutores (o seu desempenho não é igual, pois possuem menor velocidade de acesso, mas pode-se dizer que é semelhante);

b) não ser volátil (característica essencial para que o computador possua memórias rápidas e permanentes);

c) ter segurança, permitir apenas leitura de seu conteúdo por determinados programas. Há determinados programas críticos que não gostaríamos de ver infectados por vírus, por exemplo.

Todo sistema de computação utiliza uma parte do espaço de endereçamento da memória principal com memórias do tipo ROM. Os microcomputadores do tipo PC, por exemplo, vêm da fábrica com um conjunto de rotinas básicas do sistema operacional armazenadas em ROM, denominadas em conjunto de "BIOS — Basic Input Output System", ou Sistema Básico de Entrada e Saída.

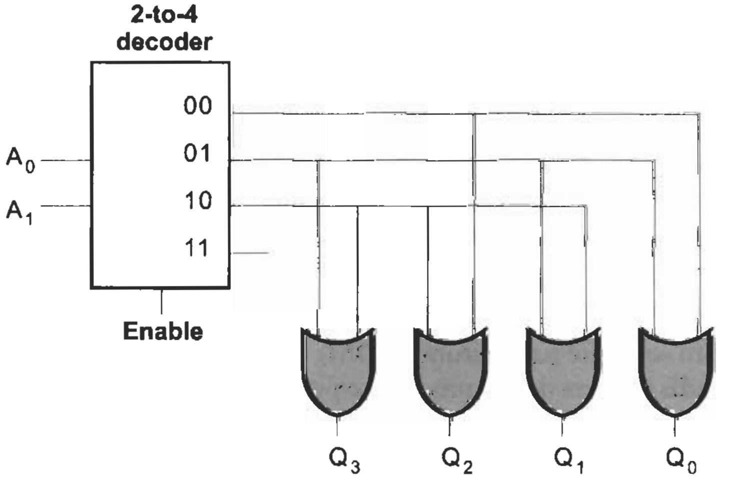
Outra aplicação importante das ROM é o armazenamento de microprogramas em memória de controle (ROM) (ver item 6.6.4.2). Como também em sistemas de controle de processos, como sistemas de injeção eletrônica de automóveis, fornos de microondas e outros eletrodomésticos controlados por computadores, assim como em jogos eletrônicos (*video games*).

As memórias ROM também sofreram uma evolução tecnológica ao longo do tempo, principalmente para torná-las mais práticas e comercialmente aceitáveis, sem perder a sua principal característica de serem memórias somente para leitura por parte dos programas aplicativos (embora, com sua evolução, elas possuam tipos que permitem a troca do seu contéudo, sempre através de processos especiais e nunca por um simples programa aplicativo).

O primeiro e original tipo é a chamada memória ROM pura (usamos este nome "pura" para diferençar de outros tipos de ROM, mais flexíveis, como veremos a seguir), que é também conhecida tecnicamente como "programada por máscara" (*mask programmed*), devido ao processo de fabricação e escrita dos bits na memória.

Nessa ROM pura, o conjunto de bits (programa especificado pelo usuário) é inserido no interior dos elementos da pastilha durante o processo de fabricação. Em inglês chama-se a isto de processo *hardwired*, pois cada bit (seja 0 ou 1, conforme o programa) é criado já na célula apropriada. Após o término da fabricação, a pastilha ROM está completa, com o programa armazenado, e nada poderá alterar o valor de qualquer de seus bits.

Por ser dessa forma, a ROM se torna um dispositivo muito mais simples do que uma RAM, necessitando apenas de um decodificador de endereços, as correspondentes linhas de barramento de saída e alguns circuitos lógicos (por exemplo, operadores OR). A Fig. 5.17 mostra um exemplo de uma memória ROM, constituída de 4 células de 4 bits cada uma, possuindo, então, 4 endereços de 2 bits cada um.



Location	Stored word
00	0101
01	1011
10	1110
11	0000

Figura 5.17 Memória ROM com 4 células de 4 bits cada.

Na Fig. 5.17 podemos observar o decodificador de endereços com uma entrada para um endereço de 2 (dois) bits, A_0 e A_1, as linhas de saída do decodificador para as 4 portas lógicas OR, S_0, S_1, S_2 e S_3, cada uma responsável pela geração de um dos 4 bits das células da memória, conforme o endereço dado.

Como exemplo, seja o endereço de entrada 01, sendo 0 na linha A_0 e 1 na linha A_1. O decodificador produzirá um sinal positivo (bit 1) na linha correspondente ao valor 01 da figura, que incidirá sobre uma das entradas das portas S_3, S_1 e S_0. A outra entrada de cada porta, bem como as duas entradas da porta S_2, terá valor nulo (bit 0), já que não incide nada da saída do decodificador. A combinação de bit 0 e bit 1 nas 3 portas aqui indicadas produzirá como saída um bit 1 em cada uma, e na restante, S_2, sairá um bit 0. Em conclusão, o valor da célula a ser lido será 1011, conforme programado na fabricação da memória.

Trata-se de um processo semelhante ao da fabricação de CDs, onde se cria primeiro uma matriz do CD desejado (no caso é uma matriz da pastilha com todos os bits inseridos), a qual tem um valor financeiro apreciável, e, em seguida, se realiza a prensagem (reprodução da matriz em cada cópia) das cópias (no caso, trata-se da criação das demais pastilhas). O método também é semelhante ao da fabricação de processadores.

Esse tipo de memória é relativamente barato se fabricado em grandes quantidades, porque, nesse caso, o custo da fabricação da máscara de programa é diluído para cada pastilha. No entanto, há certas desvantagens:

a) não há possibilidade para recuperação de qualquer erro eventual no programa. Se inserir um único bit errado na pastilha (em geral é no conjunto fabricado, devido ao processo de máscara), o lote deve ser destruído e fabricada nova partida correta. Isto pode acarretar problemas de custo do sistema;

b) o custo (que não é pequeno) da criação da máscara para inserção dos bits é o mesmo, seja para fabricar uma pastilha ou milhares delas.

Para atenuar o problema do custo fixo da máscara (matriz), desenvolveu-se uma variação daquele tipo de memória ROM pura, denominada PROM (Programmable Read Only Memory) — ROM programável. Na realidade, não se trata propriamente de ser programável, porque não é possível a reutilização da PROM (como também não se reutiliza a ROM). Nela, como nas ROM puras, somente é possível gravar os bits desejados uma única vez, porém com a diferença de que a gravação dos bits é posterior à fase de fabricação da pastilha, embora deva ser realizada por dispositivo especial.

Uma PROM é, então, fabricada "virgem" (sem qualquer bit armazenado) e depois, seja pelo usuário, seja pelo fabricante do programa ou por qualquer agente especializado, que possua a máquina específica para inserir os bits, ocorre a etapa de gravação da informação. Após o término da gravação, também não é possível efetuar qualquer alteração. Esta etapa de gravação é conhecida como processo de "queimar" a pastilha (ver item 5.6).

Trata-se de um processo semelhante ao utilizado atualmente pelos dispositivos gravadores de CD, que produzem os CD-R. Nestes também se usa um CD virgem e o gravador de CD insere os elementos de informação no CD-R, o qual é, em seguida, "queimado", inviabilizando outras gravações.

Uma ponderável diferença entre ROM e PROM reside no seu custo individual. Como já mencionado, as ROM só se tornam atraentes se fabricadas em grande quantidade, pois, neste caso, o custo fixo da matriz é dividido por uma grande quantidade de cópias. Para menores quantidades, a PROM se torna mais conveniente devido ao menor custo individual, o qual independe da quantidade (não há custo fixo de fabricação de máscara).

Posteriormente foram desenvolvidos outros dois tipos de ROM, os quais possuem uma particularidade interessante. Conquanto se mantenham somente para leitura (ROM) de programas aplicativos, durante uma execução normal, elas podem ser apagadas (através de um processo especial, que depende do tipo) e regravadas, sendo portanto reutilizáveis. São elas: EPROM (Erasable PROM) — PROM apagável, — EEPROM (Electrically ou Electronically EPROM) ou EPROM eletrônica, também chamada EAROM (Electrically Alterable ROM) e a memória Flash ou Flash-ROM. São memórias úteis no caso de aplicações que requerem muito mais leitura de dados do que escrita, sendo o caso, por exemplo, de programas de sistemas (controle do vídeo, de modems, de dispositivos de entrada/saída), onde o fabricante escreve o programa, que é intensamente lido pelos aplicativos, mas que não deve ser por eles modificado (escrito por cima). Eventualmente, o fabricante precisa modificar seu programa, criando uma nova versão, o que pode ser realizado nessas memórias.

A EPROM pode ser utilizada diversas vezes porque os dados nela armazenados podem ser apagados ao se iluminar a pastilha com luz ultravioleta (ver Fig. 5.18), a qual incide em uma janela de vidro, montada na parte superior da pastilha. O processo de apagamento, que é completo (todo o conteúdo da memória é apagado), dura em média cerca de 20 a 25 minutos.

Uma vez apagada, a pastilha pode ser reutilizada através de novo processo de "queima" de novos bits. Após esse passo, a janela de vidro costuma ser coberta para evitar um apagamento acidental.

O outro tipo, EEPROM ou EAROM, desenvolvido posteriormente, introduziu uma característica mais versátil e prática no processo de reutilização das ROM: a programação (escrita dos bits), o apagamento e a reprogramação são efetuados através de controle da UCP, isto é, por software. Uma grande vantagem desse tipo de memória é o fato de as operações poderem ser realizadas especificamente sobre um byte ou bytes determinados.

Com a EEPROM programada, as instruções nela armazenadas são retidas indefinidamente ou até que um sinal de apagamento seja captado por um sensor.

Uma boa aplicação para a EEPROM consiste em se utilizar programação das teclas de um teclado. Nesse caso, a função de cada tecla é definida em uma tabela, que reside em uma EEPROM instalada no circuito impresso normalmente localizado na parte interna do teclado. Programas aplicativos (como processadores de texto, planilhas etc.), ao serem carregados na memória do sistema para execução, armazenam na EEPROM funções de teclas específicas para o referido aplicativo. Dessa forma, as funções do teclado podem ser padronizadas para cada aplicação.

Finalmente, um outro tipo dessas memórias é denominado Flash, tendo processo de funcionamento bastante semelhante ao das EEPROM, ou seja, o conteúdo total ou parcial da memória pode ser apagado normalmente por um processo de escrita, embora nas flash o apagamento não possa ser efetuado ao nível de byte como nas EEPROM. O termo flash foi imaginado devido à elevada velocidade de apagamento dessas memórias em comparação com as antigas EPROM e EEPROM.

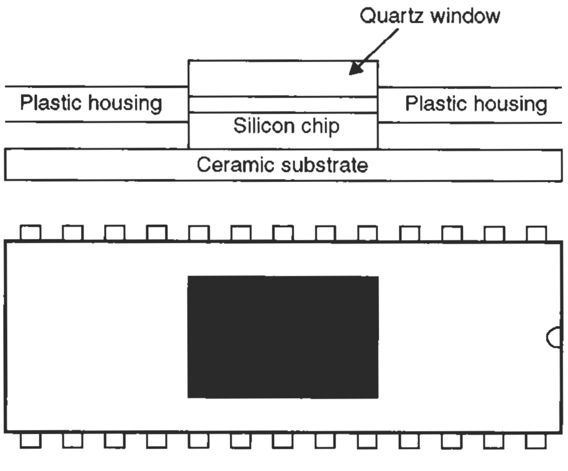


Figura 5.18 Uma pastilha de EPROM inclui uma janela transparente pela qual um feixe de luz ultravioleta pode apagar todo o seu conteúdo.

5.4 ERROS

Em todo sistema de transmissão de informação a distância (telecomunicações) há sempre a possibilidade de ocorrerem deformações ou até mesmo destruição de parte da informação transmitida (ou toda). Isso ocorre devido a interferências no meio de transmissão.

A memória principal (ou qualquer outro tipo de memória) utiliza um meio de transmissão (barramento de dados) para o trânsito da informação (palavras de dados ou instruções) entre a MP e a UCP. Esse trânsito sofre interferências que podem alterar o valor de 1 ou mais bits (de 0 para 1 ou de 1 para 0) ou até mesmo destruí-los.

Não faz parte do escopo deste livro descrever as possíveis causas da existência de erros na transmissão e armazenamento de informações na MP de um sistema de computação, mas é importante sabermos que os atuais sistemas de memória possuem mecanismos capazes de detectar e corrigir tais erros. A Fig. 5.19 sumariza *o processo básico de detecção e correção de erros, que pode ser resumido nas seguintes etapas:*

a) Os grupos de M bits de informação que serão gravados nas células da MP sofrem um processamento específico, em um dispositivo próprio para detecção de erros. Esse processamento é realizado segundo as etapas de um algoritmo determinado (A) e produz, como resultado, um conjunto de K (M) bits.

b) Serão gravados, então, em células com capacidade para armazenar M + K bits (e não apenas os M bits de informação).

c) Ao ser recuperado o valor em bits de uma determinada célula (operação de leitura), o sistema de detecção é acionado; o mesmo algoritmo inicial (A) é executado sobre os M bits de informação armazenados, obtendo-se um novo conjunto de K bits (K2).

d) Os K (M) bits armazenados são comparados com os K (K2) bits calculados, obtendo-se um entre os seguintes possíveis resultados:

1 – ambos os conjuntos de K bits têm o mesmo valor, o que significa ausência de erros. Neste caso, os M bits da célula desejada são transmitidos;

2 – os conjuntos são diferentes, concluindo-se pela existência de erro no bloco de M bits. O erro pode ser corrigido ou não, dependendo de como o sistema foi projetado.

O processo de correção de erros, denominado comumente ECC (Error Correction Code ou Código de Correção de Erro) baseia-se no código utilizado para constituir os K bits adicionais de cada célula. Em geral, este método é utilizado em computadores com aplicações mais sensíveis, tais como um servidor de arquivos ou servidor de rede. Eles podem detectar a ocorrência de erros em 1 ou mais bits e corrigir 1 bit errado. Ou seja, se for detectada a ocorrência de erro em 1 bit, este será identificado pelo código e naturalmente corrigido; porém, se forem detectados mais bits errados, então, o ECC somente indica o erro sem poder identificar (e corrigir) os bits errados.

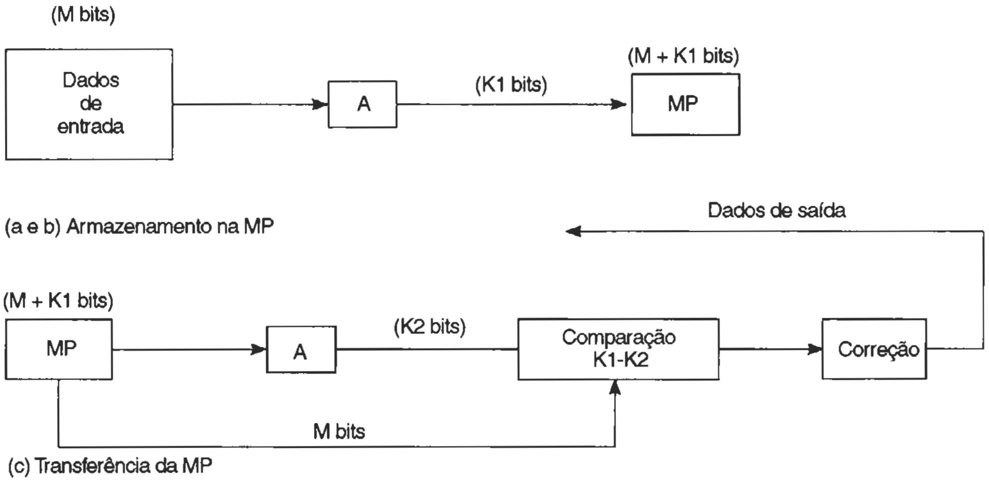


Figura 5.19 Processo básico de detecção/correção de erros.

5.5 MEMÓRIA CACHE

No item 5.2.2 foi apresentada uma breve explicação sobre o conceito de memória cache, bem como dados de seu desempenho e características de modo a situá-la adequadamente na estrutura piramidal de uma hierarquia de memória.

Este item pretende detalhar um pouco mais os conceitos e técnicas que norteiam a fabricação e o uso de memórias cache em sistemas de computação modernos.

5.5.1 Conceitos

Para que seja possível entender perfeitamente o sentido da criação e desenvolvimento das memórias cache e de sua crescente e permanente utilização nos sistemas de computação, devem ser conhecidos dois conceitos relacionados com o funcionamento e a utilidade dessas memórias: o conceito de diferença de velocidade UCP/ MP e o conceito de localidade.

5.5.1.1 Diferença de Velocidade UCP/MP

Não se trata propriamente de um conceito, mas sim de uma constatação inevitável. Parte do problema de limitação de velocidade do processador, que qualquer projetista de sistemas de computação enfrenta, refere-se à diferença de velocidade entre o ciclo de tempo da UCP e o ciclo de tempo da memória principal. Ou seja, a MP transfere bits para a UCP em velocidades sempre inferiores às que a UCP pode receber e operar os dados, o que acarreta, muitas vezes, a necessidade de acrescentar-se um tempo de espera para a UCP (*wait state* — estado de espera).

Se o processador (UCP) precisa esperar pelos bits enviados da MP, isso se faz em períodos fixos de tempo, um estado de relógio (ver Cap. 6). Cada período destes é chamado de estado de espera (*wait state*). O número de estados de espera inseridos para que a transferência de dados se concretize depende da relação entre a velocidade da UCP e a da MP.

Se todos os circuitos da UCP e da MP fossem fabricados com elemento de mesma tecnologia, este problema deixaria de existir e não estaríamos aqui explicando o que é e para que serve uma memória cache.

O problema de diferença de velocidade se torna difícil de solucionar apenas com melhorias no desempenho das MP, devido a fatores de custo e tecnologia. Já foi anteriormente mencionado que, enquanto o desempenho dos microprocessadores, por exemplo, vem dobrando a cada 18 a 24 meses, o mesmo não acontece com a velocidade de transferência (tempo de acesso) das memórias DRAM (RAM dinâmicas), largamente utilizadas como MP, que vem aumentando pouco de ano para ano (cerca de 10%).

Apesar de a tecnologia para aumentar a velocidade das MP ser bem conhecida, fazer isso neste momento penalizaria consideravelmente o sistema em custo, pois memórias rápidas são muito caras, como é o caso das memórias SRAM.

5.5.1.2 Conceito de Localidade

Basicamente e de modo simplista, podemos definir o conceito de localidade como o fenômeno relacionado com o modo pelo qual os programas em média são escritos e executados pela UCP. Pesquisas e amostragens realizadas por diversos cientistas concluíram que a execução dos programas se realiza, na média, em pequenos grupos de instruções. Este princípio, aliás, não é aplicado apenas em memórias cache, mas deu origem, como já observamos anteriormente, ao desenvolvimento da hieraquia de memória, com a implementação de diversos tipos diferentes de memória em um sistema de computação.

Continuando, esse fato pode ser decomposto em duas facetas do princípio da localidade: *espacial* e *temporal.*

Assim, os programas não são executados de modo que a MP seja acessada randomicamente, como seu nome sugere (RAM). Se um programa acessa uma palavra da memória, há uma boa probabilidade de que ele em breve acesse a mesma palavra novamente. Este é o princípio da *localidade temporal.* E se ele acessa uma palavra da memória, há uma boa probabilidade de que o programa acesse proximamente uma palavra subseqüente ou de endereço adjacente àquela palavra que ele acabou de acessar. É o princípio da *localidade espacial.*

A Fig. 5.20 apresenta um exemplo do princípio da localidade espacial.

No exemplo dessa figura, um certo programa pode ser constituído de um grupo de instruções iniciais, realizadas em seqüência (parte 1), de 2 loops (loop 1 e loop 2) de uma sub-rotina chamada dentro do loop 1, o que significa que ela será repetida diversas vezes, e do resto do código (parte 2 e parte 3).

O que acontece é que cada parte do programa (parte 1, loop 1, sub-rotina, loop 2, parte 2 e parte 3) é realizada separadamente, isto é, durante um tempo a UCP somente acessa o grupo de instrução da parte 1,

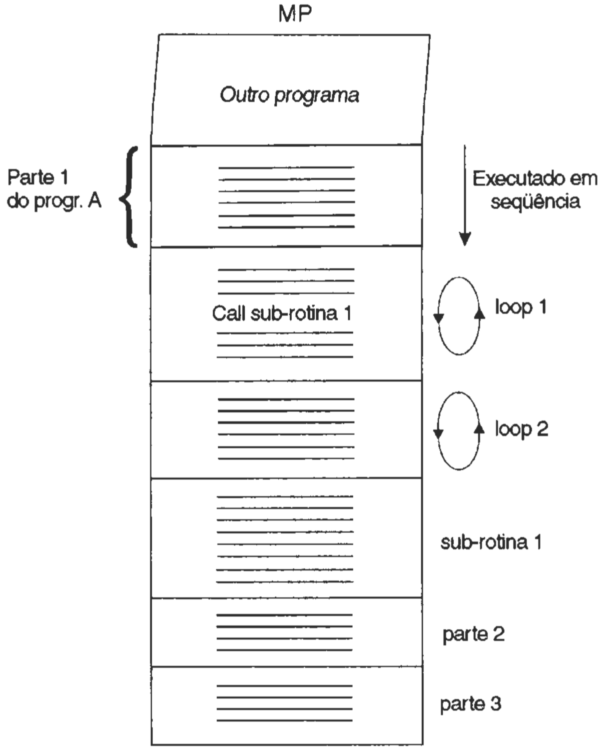


Figura 5.20 Um programa em execução com várias partes (exemplo do princípio de localidade especial).

depois se dedica ao loop 1 e, neste, diversas vezes salta para a área da sub-rotina e acessa somente seu código, e assim por diante. Ou seja, o programa não salta indiscriminadamente da primeira instrução para uma no meio do programa, depois para outra no final, retornando para o início etc.

Assim, seria uma boa idéia tirar vantagem daqueles princípios de localidade, colocando a parte repetitiva de um pedaço do programa em uma memória bem rápida e deixando o restante do programa, que não está sendo utilizado no momento, na memória mais lenta e de maior capacidade, porém mais barata.

5.5.2 Utilização da Memória Cache

Como aproveitar, então, os princípios da localidade? Conforme já visto, o projetista do sistema cria um elemento de memória intermediário entre a UCP e MP, como mostrado na Fig. 5.21. Este elemento de memória, denominado *memória cache*, deve possuir elevada velocidade de transferência e um tamanho capaz de armazenar partes de um programa, suficientemente grandes para obter o máximo rendimento dos princípios da localidade e suficientemente pequenas para não elevar em excesso o custo do sistema de computação.

Com a inclusão da memória cache podemos enumerar, de modo simplista, o funcionamento do sistema:

- Sempre que a UCP vai buscar uma nova instrução (ou dado), após a busca inicial, ela acessa a memória cache.
- Se a instrução (ou dado) estiver na cache, chama-se de *acerto* (ou *hit*), ela é transferida em alta velocidade (compatível com a da UCP).
- Se a instrução (ou dado) não estiver na cache, chama-se de *falta* (ou *miss*), então o sistema está programado para interromper a execução do programa e transferir a instrução desejada da MP para a cache. Só que essa transferência não é somente da instrução desejada, mas dela e de um grupo subseqüente, na pressuposição de que as instruções do grupo serão requeridas pela UCP em seguida (princípio da localidade) e, portanto, já estarão na cache quando necessário (acertos).

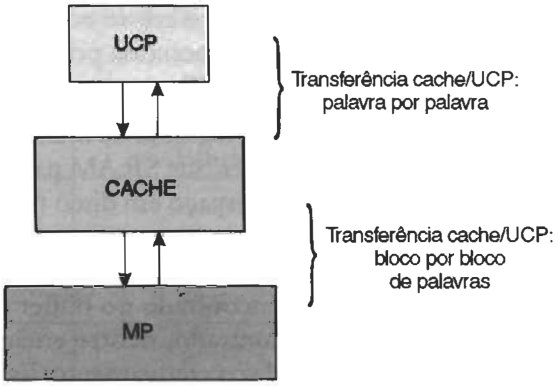

UCP
CACHE
MP
Transferência cache/UCP:
palavra por palavra
Transferência cache/UCP:
bloco por bloco
de palavras

Figura 5.21 Organização para transferência de informações entre UCP/Cache/MP.

A Fig. 5.21 bem demonstra como varia a unidade de transferência nos dois casos: palavra por palavra, no caso da UCP/cache (como seria entre UCP/MP se não existisse a cache), e bloco a bloço de palavras, quando for transferência cache/MP.

Para haver realmente algum aumento de desempenho de um sistema de computação, com a inclusão da memória cache, é necessário que haja muito mais acertos do que faltas. Isto é, a memória cache somente é produtiva (aumento de desempenho do sistema) se a UCP puder encontrar uma quantidade apreciável de palavras na cache, suficientemente grande para sobrepujar as eventuais perdas de tempo com faltas que redundam em transferência de um bloco de palavras da MP para a cache, além da efetiva transferência da instrução desejada, agora já existente na cache (depois da transferência do bloco que a contém). A taxa de acertos (*hits*) mais comum varia entre 80% e 99%, isto é, na pior das hipóteses, em cerca de 80% das vezes em que a UCP procura uma instrução ou dado ela os encontra na memória cache.

A Fig. 5.22 mostra um processador com barramento único (ônibus) e memória cache incluída.

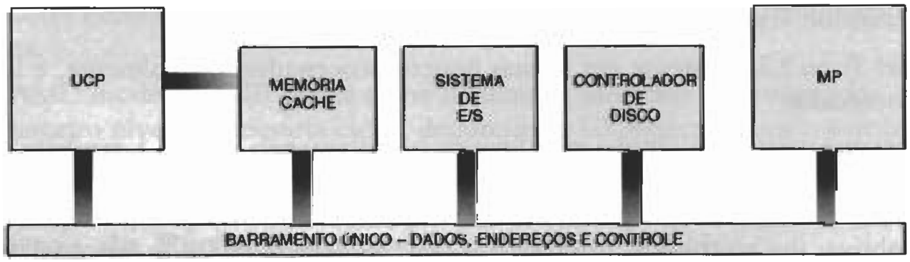

UCP
MEMÓRIA CACHE
SISTEMA DE E/S
CONTROLADOR DE DISCO
MP
BARRAMENTO ÚNICO – DADOS, ENDEREÇOS E CONTROLE

Figura 5.22 Exemplo de um sistema de computação (microcomputador) com utilização de memória cache em um barramento único.

5.5.3 Tipos de Memória Cache

A importância das memórias cache nos sistemas de computação vem se tornando inquestionável e, a cada novo sistema, ela se torna mais imprescindível para um correto e eficaz desempenho dos referidos sistemas. Esta importância tem se traduzido, entre outros elementos, no desenvolvimento e criação de diferentes tipos de cache.

Em princípio, pode-se definir dois tipos básicos de emprego de cache:

– na relação UCP/MP (Cache de RAM ou RAM Cache) e

– na relação MP/Discos (Cache de Disco ou Disk Cache).

O primeiro tipo, cache de RAM ou cache para a MP, refere-se ao conceito exposto nos itens anteriores, onde a memória cache é utilizada para substituir o uso da memória principal (ou RAM) pela UCP, acelerando o processo de transferência de dados desejados pela UCP.

No segundo tipo, cache de disco, o sistema funciona segundo os mesmos princípios da cache de memória RAM, porém, em vez de utilizar a memória de alta velocidade SRAM para servir de cache, o sistema usa uma parte da memória principal, DRAM, como se fosse um espaço em disco (vale-se de uma parte da RAM como *buffer*). Desse modo, quando um programa requer um dado que esteja armazenado em disco, o sistema verifica em primeiro lugar se o dado está no espaço reservado na memória RAM e que simula o espaço em disco; da mesma forma que na cache de RAM, se o dado for encontrado no buffer da memória (que simula o espaço em disco) haverá um acerto (*hit*), e se lá não for encontrado, ocorre então uma falta (*miss*) e, nesse caso, o sistema interrompe seu processamento para acessar o disco efetivamente, localizar e transferir o dado desejado e mais um bloco de dados subseqüentes, de modo semelhante ao da cache de RAM.

Assim como no caso de cache de RAM, a cache de disco pode aumentar excepcionalmente o desempenho do sistema visto que o acesso à memória RAM (faixa de nanossegundos) é milhares de vezes mais rápido que o acesso ao disco (faixa de milissegundos).

5.5.3.1 Níveis de Cache de Memória RAM

No que se refere às memórias cache de RAM (*RAM Cache*), o aumento crescente da velocidade das UCP e o compromisso de não se aumentar demasiado o custo das memórias cache (o que ocorreria inevitavelmente com o aumento de sua capacidade, conduziram os projetistas e cientistas a desenvolverem caches com diferentes características de velocidade e capacidade, formando também um sistema hierárquico em níveis ou camadas, a exemplo do próprio sistema global de memória.

Assim é que, atualmente, os fabricantes e montadores de sistemas estabeleceram dois e até três diferentes níveis de memória cache, todos constituídos de memórias SRAM, porém cada um com diferentes características. Estes níveis/camadas são:

- Nível 1 (*Level 1*) ou L1, sempre localizada no interior do processador;
- Nível 2 (*Level 2*) ou L2, sendo localizada, em geral, na placa-mãe do computador, ou seja, externa ao processador. Porém, têm sido lançados processadores onde a cache L2 está localizada no interior da pastilha do processador, separada deste.
- Nível 3 (*Level 3*) ou L3, existente em apenas poucos processadores, atualmente, e localizada externamente ao processador.

Os computadores que possuem múltiplas memórias cache iniciam pela cache L1, também denominada cache primária. Esta memória, constituída de elementos com tecnologia SRAM e com velocidade de acesso igual à do processador, é, por esta razão, a que possui menor tempo de acesso. Sempre que o processador buscar um dado/instrução e obtiver um acerto (*hit*), a velocidade de transferência será a do processador, com um tempo quase instantâneo.

O funcionamento do sistema é sempre o mesmo de casos anteriores: a memória de nível mais baixo é a que tem maior velocidade (e também maior custo) e menor capacidade. A UCP sempre procura o dado/instrução na memória de menor nível; não encontrando na L1, buscará na L2 (se houver uma); desta para a memória DRAM e, finalmente, não encontrando na DRAM irá buscar o dado/instrução no disco.

Apesar de as memórias cache L1 serem sempre internas, isto é, construídas com os mesmos elementos do processador, pode haver diferenças de arquitetura entre elas, que modificam seu desempenho. É o caso, por exemplo, de um processador com cache de maior tamanho que outro ou se um processador possui ou não a cache dividida, cache L1 de instruções e cache L1 de dados; a maior capacidade e/ou a divisão da cache interna, L1, podem tornar um processador mais rápido que outro, até mesmo se esse outro possuir um processador mais rápido. Por exemplo, o processador Pentium MMX possui cache dividida de 32KB (16KB para dados e 16KB para instruções, sendo normalmente especificado assim: cache 16 + 16), enquanto o Pentium I original possui cache de 16KB (8 + 8), sendo, assim, o primeiro mais rápido que o Pentium original devido à diferença de capacidade da cache L1.

A Tabela 5.1 apresenta alguns processadores e suas memórias cache L1.

Tabela 5.1

Processadores	Fabricante	Tamanho
486	Intel	8KB, unificado
C6	Cyrix	64KB, dividido
K5	AMD	24KB, dividido
K7	AMD	128KB, dividido
Pentium	Intel	16KB, dividido
Pentium MMX	Intel	32KB, dividido
Pentium PRO	Intel	16KB, dividido
Power PC601	Motorola/IBM	32KB
Pentium III	Intel	32KB, dividido

A memória cache L2 ou cache externa, ou ainda denominada cache secundária, é usualmente instalada na placa-mãe do sistema, externamente ao processador, mas ainda constituída de elementos com tecnologia SRAM. Naturalmente, possui menor velocidade de transferência do que a cache L1, visto que opera na velocidade do barramento de dados, da placa-mãe, sempre mais lento que a velocidade do processador. No entanto, possui uma capacidade bem maior de armazenamento que as memórias cache L1, em uma faixa de 64KB a 4MB (é possível que ao ler este texto o leitor já tenha tido informação sobre valores ainda maiores, o que é perfeitamente natural, em face da velocidade de avanço da tecnologia desses componentes).

Apesar de a cache L2 ser também conhecida como cache externa, alguns processadores têm sido fabricados com a cache L2 instalada internamente na pastilha do processador, embora em um invólucro separado deste, mas não podendo mais ser chamada de cache externa. É o caso, por exemplo, do Pentium Pro, Pentium II e III. Já os processadores Pentium e Pentium MMX possuem cache L2 externa, sendo esta naturalmente uma memória mais lenta.

O processador AMD modelo K6-III possui cache L2 interna, integrada ao processador, e foi desenvolvido para permitir um terceiro nível de memória cache, denominado L3, externo, com controlador na placa-mãe. É, até o momento em que este livro está sendo escrito, o único processador com esta arquitetura (cache L3).

5.5.4 Elementos de Projeto de uma Memória Cache

Para se efetivar o projeto e implementação de uma memória cache deve-se decidir entre várias alternativas tecnológicas, atualmente disponíveis, as quais podem ser agrupadas por função:

- definição do tamanho das memórias cache, L1 e L2;
- função de mapeamento de dados MP/cache;
- algoritmos de substituição de dados na cache;
- política de escrita pela cache.

5.5.4.1 Tamanho da Memória Cache

Já foi explicado que uma memória cache só é produtiva, isto é, tem o desempenho do sistema bastante melhorado com seu uso se, durante a execução de um programa, ocorrerem muito mais acertos (o dado requerido pela UCP se encontra na cache) do que faltas (o dado requerido pela UCP não se encontra na cache *e deve, então, ser transferido da memória DRAM (MP) para a cache (seja L1, seja L2), para desta ir para a* UCP), e se isso for obtido através do emprego de uma memória cache de tamanho relativamente pequeno, para que o custo do sistema seja baixo.

A definição da faixa de tamanho adequada para a urna cache depende de uma série de fatores, específicos de um certo sistema, tais como:

- tamanho da memória principal;
- relação acertos/faltas;
- tempo de acesso da MP;
- custo médio por bit, da MP, e da memória cache L1 ou L2;
- tempo de acesso da cache L1 ou L2;
- natureza do programa em execução (princípio da localidade).

Vários estudos têm sido realizados a respeito [STRE83], podendo-se considerar uma faixa entre 8 e 128 Kbytes para memórias cache do tipo L1 (cache primária ou interna) e de 64K até 4 Mbytes para memórias cache do tipo L2 (cache secundária). A Tabela 5.1 apresenta alguns exemplos de memórias cache L1 e seus respectivos tamanhos.

5.5.4.2 Mapeamento de Dados MP/Cache

Para entender melhor o sentido desta função como elemento de projeto de memória cache, consideremos que a memória RAM (MP) consiste em um conjunto seqüencial de $N = 2^E$ palavras endereçáveis (células), cada uma possuindo um único e unívoco endereço com E bits; as palavras (células) são dispostas desde a palavra de endereço 0 até a palavra de endereço (N–1) (ver Fig. 5.23).

Consideremos ainda que a memória RAM (MP) está dividida em um conjunto de B blocos (numerados de 0 a B-1), cada um constituído de K células (palavras), sendo $B = N/K$ ou $B = 2^E/K$. A memória cache constituída de Q linhas de dados, cada uma podendo armazenar K palavras (células). Q é muito menor que B para que a cache não seja muito cara. Normalmente os sistemas são organizados com uma memória cache (L2) de cerca de 512KB, enquanto a memória RAM (MP) tem um valor típico da ordem de 64MB.

As explicações a seguir, bem como os exemplos dados para os diversos métodos de mapeamento, estão considerando precipuamente a arquitetura e características das memórias cache secundárias, L2. No entanto, a quase totalidade das informações dadas também se aplicam às memórias L1, com exceções típicas de seu formato e arquitetura.

Observemos, então, esta extraordinária tecnologia de uso da memória cache, a qual (cache L2) possui uma capacidade menor que 1% da capacidade da memória RAM e consegue obter 90% a 95% de taxa de acertos (*hits*). É por essa razão que os fabricantes hoje em dia utilizam intensamente e cada vez mais o conceito de cache, inclusive criando mais de um tipo, L1 e L2 (e já vimos que há fabricantes criando a L3). Naturalmente, esta enorme eficácia é possível graças ao conhecimento do princípio da localidade, exposto anteriormente. Para recapitularmos este princípio utilizemos um simples exemplo, através de um pequeno *loop*, dos inúmeros que qualquer programa tem.

Suponhamos um trecho de um programa do tipo:

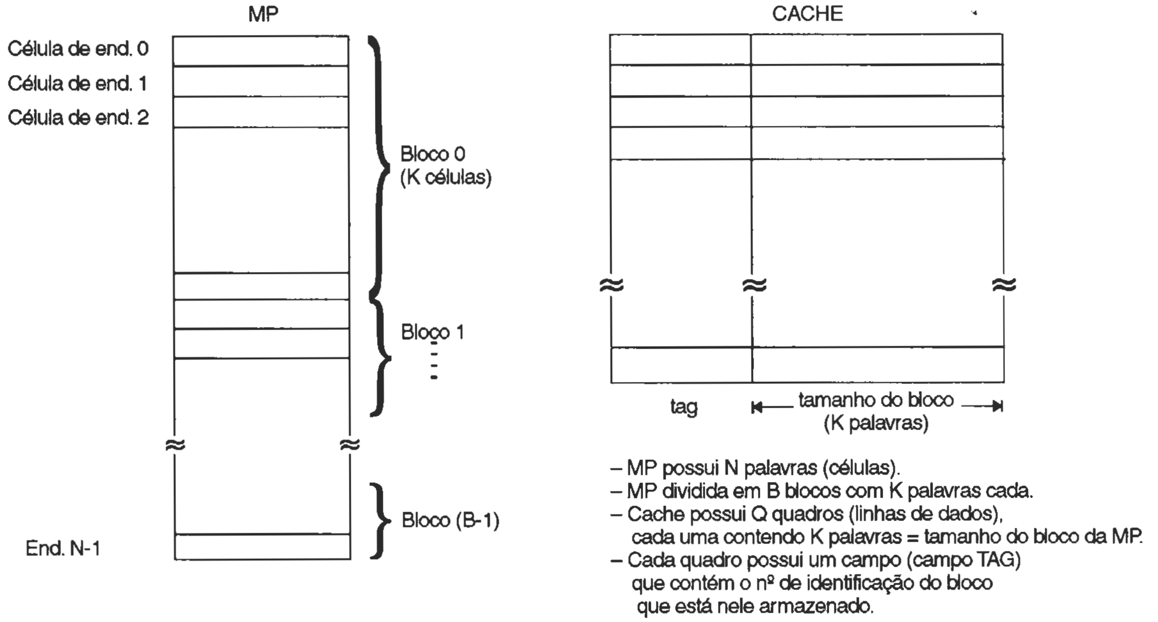
```
   Índice = 1000
   Enquanto Índice diferente de zero
   Iniciar
      X (Índice) = A (Índice) + B (Índice)
      Índice = Índice − 1
Terminar
```

Neste exemplo, consideremos que a memória cache é capaz de armazenar este simples trecho do programa. Isto significa que em 999 vezes das 1000, o sistema obterá um acerto (*hit*), ou seja, uma taxa de acerto da ordem de 99,9%, mesmo considerando-se que sua capacidade seja muito menor que a da memória RAM.

A cada instante, a memória cache possui um conjunto de blocos de MP armazenados em suas *linhas,* com os dados que a UCP deve precisar (para haver acertos e não falhar). Porém, como Q < B (há mais blocos que linhas), não é possível uma *linha* da cache estar dedicada a armazenar um específico bloco da MP, ou seja, uma linha é usada por mais de um bloco e, por conseguinte, é preciso identificar, em cada instante, qual o específico bloco que está armazenado na específica linha da cache. Para isso, cada linha tem um campo (além daquele para armazenar as K palavras), denominado etiqueta (*tag*), que contém a identificação do bloco e que, como veremos a seguir, faz parte dos E bits do endereço completo da MP.

Para efetuar a transferência de um bloco da MP para uma específica linha da memória cache, pode-se escolher uma das três alternativas atualmente disponíveis:

- Mapeamento direto.
- Mapeamento associativo.
- Mapeamento associativo por conjuntos.



- MP possui N palavras (células).
- MP dividida em B blocos com K palavras cada.
- Cache possui Q quadros (linhas de dados), cada uma contendo K palavras = tamanho do bloco da MP.
- Cada quadro possui um campo (campo TAG) que contém o nº de identificação do bloco que está nele armazenado.

Figura 5.23 Estrutura entre MP e cache para transferência de dados.

Mapeamento Direto

Por esta técnica, cada bloco da MP tem uma linha da cache previamente definida para ser armazenada. Como há mais blocos do que linhas da cache, isto significa que muitos blocos irão ser destinados a uma mesma linha, sendo preciso definir a regra a ser seguida para a escolha da linha específica de cada bloco.

Vamos exemplificar considerando uma memória principal com espaço de endereçamento de 4G palavras (células), tendo cada uma um endereço com 32 bits ($2^{32} = 4G$), como acontece nos processadores Intel Pentium ou AMD K6, K7, Motorola 68030 e outros.

Para exemplificar vamos assumir que a cache associada a esta memória RAM (MP) possui um tamanho correspondente a 64 Kbytes, divididos em 1024 (2^{10}) linhas, com 64 bytes de dados cada uma (64 palavras de dados).

A memória principal será, então, dividida em 64M blocos de 64 bytes cada (64 bytes é a mesma quantidade de bytes do bloco de dados da memória principal e da linha da cache).

Relembrando as informações do item anterior, vemos que a MP possui N palavras (células), divididas em B blocos e que a cache possui K linhas de 64 bytes cada. Sendo B = N/K, então B = 4G/64K = 64M blocos, numerados de bloco 0 até bloco 64M − 1.

Como há 64M blocos na MP e 1024 linhas na cache, então cada linha deverá acomodar (um de cada vez, é claro) 65.536 blocos (64K = 2^{16}). A Fig. 5.24 mostra, com dados, o exemplo que estamos descrevendo.

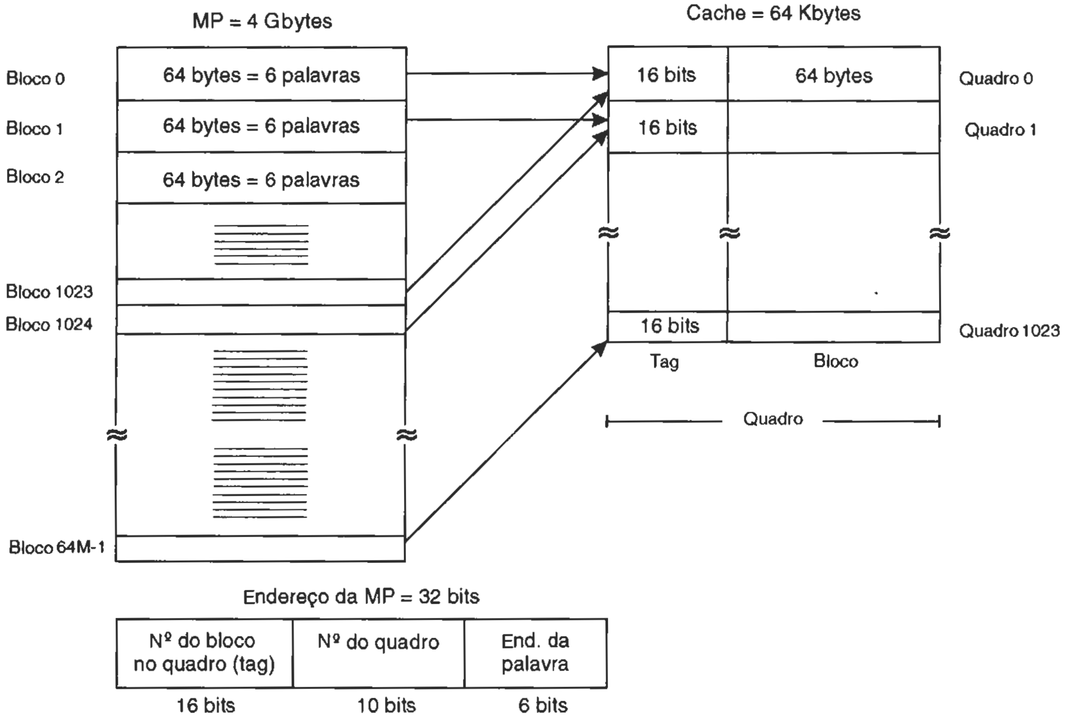

MP = 4 Gbytes
Cache = 64 Kbytes
Bloco 0
Bloco 1
Bloco 2
64 bytes = 6 palavras
64 bytes = 6 palavras
64 bytes = 6 palavras
Bloco 1023
Bloco 1024
Bloco 64M-1
16 bits
64 bytes
Quadro 0
16 bits
Quadro 1
16 bits
Quadro 1023
Tag
Bloco
Quadro
Endereço da MP = 32 bits
Nº do bloco no quadro (tag)
Nº do quadro
End. da palavra
16 bits
10 bits
6 bits

Figura 5.24 Memória cache com mapeamento direto.

Para definir quais blocos da MP serão alocados a uma linha específica, pode-se efetuar um simples cálculo usando-se aritmética de módulo (resto da divisão). Assim, para definir a linha de cada bloco, seja:

EQ = endereço da linha da cache (desde linha 0 até linha 1024 − 1)
E = endereço da RAM (MP), que vai de end. 0 até end. 4G − 1
Q = quantidade de linhas da cache, sendo no caso Q = 1024
e, então EQ = E módulo Q.

No exemplo da Fig. 5.24, Q = 1024 e, portanto, EQ = E módulo 1024, e teríamos o seguinte mapeamento previamente definido:

- para a linha 0 estarão destinados os blocos 0, 1024, 2048, 3072 e assim por diante, num total de 64K blocos;
- para a linha 1 estarão destinados os blocos 1, 1025, 2049, 3073 e assim por diante;
- e assim sucessivamente até a linha 1023, que receberá o bloco 1023, 2047, até o bloco 64M-1.

Cada endereço da memória principal pode ser dividido nos seguintes elementos (ver Fig. 5.24):

- 6 bits menos significativos: indicam a palavra (byte) de dados desejada pela UCP. Como no bloco B e na linha Q há 64 palavras (correspondentes a 64 células da memória RAM (MP), então, sendo $2^6 = 64$, cada endereço de palavra terá 6 bits);
- 10 bits do meio: indicam o endereço da linha da cache, porque $2^{10} = 1024$, e há 1024 linhas;

- 16 bits mais significativos: indicam qual o bloco, entre os 64K blocos alocados naquela linha, é o desejado. Isso porque $2^{16} = 64K$.

Desse modo, cada bloco estará *diretamente mapeado* a uma linha específica da cache. Na prática, uma operação de leitura, por exemplo, funciona da seguinte forma (ver exemplo da Fig. 5.25):

1) A UCP apresenta o endereço de 32 bits ao circuito de controle da cache, que inicia a identificação de seus campos para definir primeiramente se a palavra desejada está na cache ou não. Para exemplificar, vamos considerar o endereço binário 00000000000001000000011001001000. Para efeitos de processamento pelo sistema de controle da cache, este endereço será dividido em 3 partes, conforme já mostrado na Fig. 5.24, da esquerda para a direita, a saber:

   parte 1 (mais à esquerda, com 16 bits) = 0000000000000100

   parte 2 (meio, com 10 bits) = 0000011001

   parte 3 (mais à direita, com 6 bits) = 001000

2) Os 10 bits centrais são examinados (ver Fig. 5.25) e seu valor indica que se trata da linha $11001_2 = 25_{10}$. Resta verificar se o bloco solicitado é o que se encontra armazenado no quadro 25 ou se lá está um outro bloco.

3) O controlador da cache examina por comparação se o valor do campo tag do endereço — no caso, o valor é 4_{10} — é igual ao do campo tag da linha. No exemplo dado, eles são iguais.

4) Em seguida, é acessada a palavra 8_{10} (últimos 6 bits do endereço) e transferida para a UCP.

5) Se os valores dos campos tag, do endereço e da linha da cache não fossem iguais, isso significaria que o bloco desejado não se encontrava armazenado na cache e, portanto, deveria ser transferido da MP para o quadro 25, substituindo o atual bloco para, em seguida, a palavra (o byte) requerida — byte 8 — ser transferida para a UCP pelo barramento de dados.

Para tanto, os 26 bits mais significativos do endereço (16 bits do campo tag mais 10 bits do campo quadro) seriam utilizados como endereço do bloco desejado, pois $2^{26} = 64M$, ou seja, cada um dos 64M blocos tem um endereço com 26 bits.

A técnica de mapeamento direto é, sem dúvida, simples e de baixo custo de implementação, além de não acarretar sensíveis atrasos de processamento dos endereços. O seu problema consiste justamente na fixação da localização para os blocos (no exemplo dado, 65.536 blocos estão destinados a uma linha, o que indica que somente 1 de cada vez pode estar lá armazenado).

Se, por exemplo, durante a execução de um programa um dado código fizer repetidas referências (acessos) a palavras situadas em blocos alocados na mesma linha, então haverá necessidade de sucessivas idas à MP para substituição de blocos (muitas faltas) e a relação acerto/faltas será baixa, com a conseqüente redução de desempenho do sistema. (No exemplo da Fig. 5.24 poderíamos ter referências seguidas a uma palavra do bloco 0 e a outra palavra do bloco 1024, como também, por coincidência, outro acesso subseqüente a uma palavra do bloco 2048, todos destinados à mesma linha da cache.)

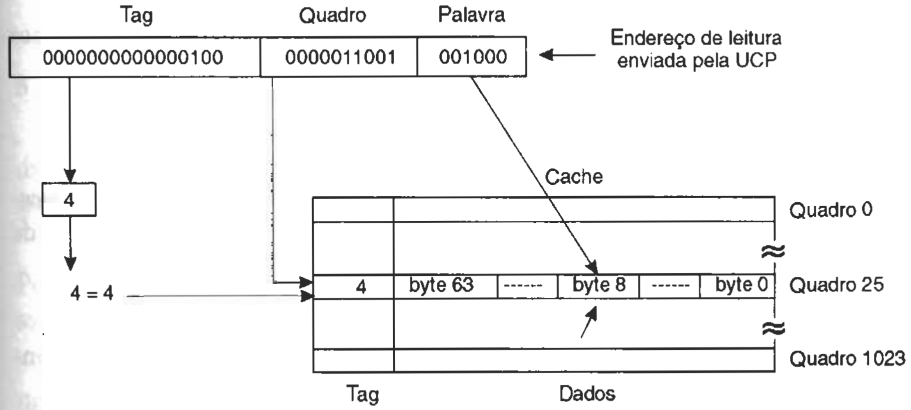


Figura 5.25 Exemplo de operação de leitura em memória cache com mapeamento direto.

Mapeamento Associativo

Nesta técnica (ver Fig. 5.26), os blocos não têm uma linha fixada previamente para seu armazenamento. Se for verificado que o bloco não está armazenado em nenhuma linha da cache, então o bloco será transferido para a cache, substituindo um bloco já armazenado nela. Que bloco deverá ser substituído é um problema a ser resolvido de acordo com o algoritmo de substituição adotado.

O endereço da MP é, neste caso, dividido apenas em duas partes:

- os mesmos 6 bits menos significativos, para indicar a palavra desejada pela UCP; e
- os 26 bits restantes, que indicam o endereço do bloco desejado.

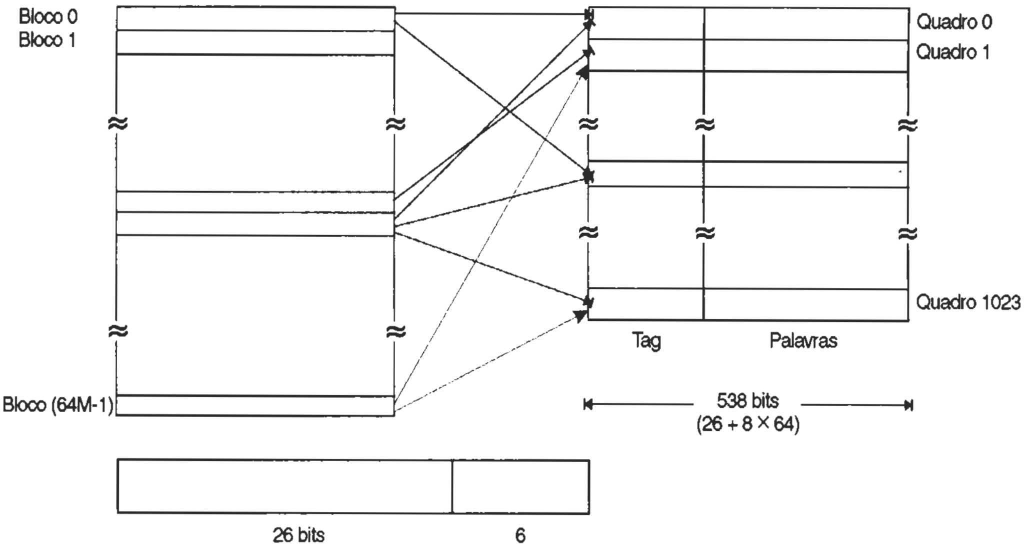


Figura 5.26 Memória cache com mapeamento associativo.

A diferença é que o campo tag de cada linha da cache tem agora 26 bits de tamanho, armazenando o endereço completo do bloco armazenado na respectiva linha.

Sempre que a UCP realizar um acesso, o controlador da cache deve examinar e comparar os 26 bits do endereço de bloco com o valor (de 26 bits também) armazenado no campo tag de todas as 1024 linhas para verificar se o bloco desejado está na cache ou não.

Caso afirmativo, o byte da palavra desejada é transferido, e caso contrário, o endereço do bloco é usado para chegar à MP e trazê-lo para a cache, substituindo um bloco existente, conforme o algoritmo de substituição escolhido.

Embora esta técnica evite a fixação dos blocos às linhas, por outro lado acarreta a necessidade de uma lógica complexa para, rapidamente, examinar cada campo tag de todas as linhas da cache.

Mapeamento Associativo por Conjuntos

Esta técnica tenta resolver o problema de conflito de blocos em uma mesma linha (da técnica de *mapeamento direto*) e o problema de técnica de *mapeamento associativo*, de exaustiva busca e comparação do campo tag de toda a memória cache.

É, pois, um compromisso entre ambas as técnicas anteriores.

A técnica se resume em organizar as linhas da cache em grupos, denominados conjuntos. Dentro do conjunto, as linhas são complementarmente associativas, como na técnica anterior.

Genericamente, a cache é dividida em C conjuntos de D linhas, de modo que:

$Q = C \times D$

$K = E$ módulo C

sendo K = endereço da linha no conjunto.

O algoritmo estabelece que:

a) o endereço da memória principal é dividido da seguinte forma:

Tag	**Número do conjunto**	**Endereço da palavra**
17 bits	**9 bits**	**6 bits**

b) ao se iniciar uma operação de leitura, por exemplo, o controlador da cache interpreta primeiramente os bits do campo de conjuntos para identificar qual o conjunto desejado (vamos considerar a opção mais comum, onde cada conjunto possui duas linhas). Em seguida, o sistema compara, no conjunto encontrado, o valor do campo tag do endereço com o valor do campo tag de cada linha do conjunto encontrado.

A Fig. 5.27 mostra o esquema da técnica, considerando-se a mesma MP (4G palavras) e cache (1024 linhas) anteriores.

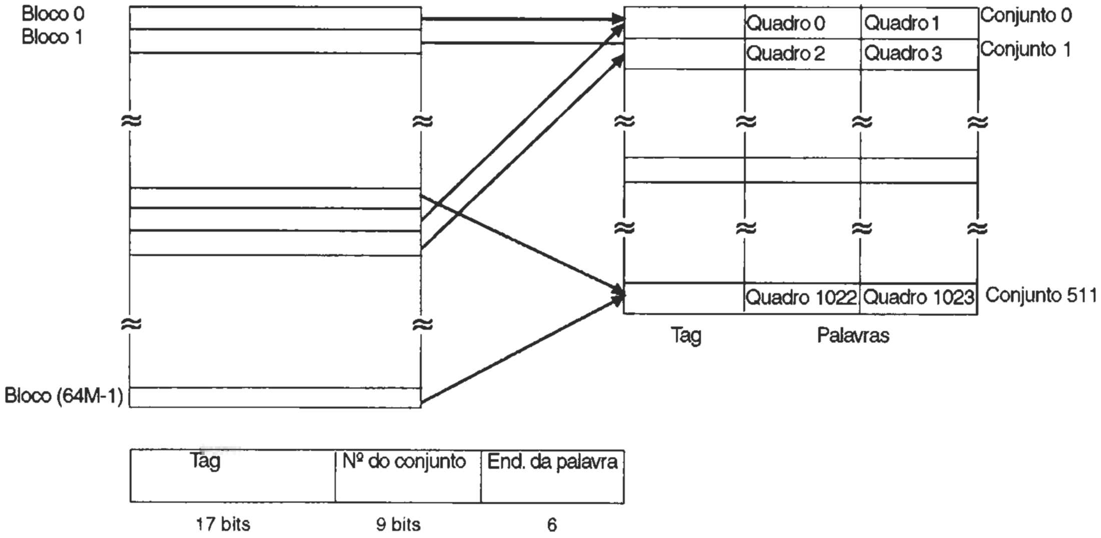

Bloco 0
Bloco 1
Bloco (64M-1)
Quadro 0
Quadro 1
Conjunto 0
Quadro 2
Quadro 3
Conjunto 1
Quadro 1022
Quadro 1023
Conjunto 511
Tag
Palavras
Tag
Nº do conjunto
End. da palavra
17 bits
9 bits
6

Figura 5.27 Memória cache com mapeamento associativo por conjunto.

5.5.4.3 Algoritmos de Substituição de Dados na Cache

O problema se resume em definir qual dos blocos atualmente armazenados na cache deve ser retirado para dar lugar a um novo bloco que está sendo transferido (é bom lembrar que isto é necessário porque todos os quadros da cache estão sempre ocupados, visto que $Q << B$).

Dependendo de qual técnica de mapeamento se esteja usando, pode-se ter algumas opções de algoritmos. Por exemplo, se o método de mapeamento adotado foi o *direto,* então não há o que definir, pois, neste caso, somente há uma única linha possível para um dado bloco (ver descrição do método).

No entanto, para os outros dois métodos de mapeamento — *associativo* e *associativo por conjunto* — pode-se optar por um dos seguintes algoritmos.

O que não é usado há mais tempo (LRU — Least Recently Used) — o sistema escolhe para ser substituído o bloco que está há mais tempo sem ser utilizado. Ou seja, trata-se do bloco que a UCP não usa há mais tempo.

Fila, ou seja, o primeiro a chegar é o primeiro a ser atendido (F1F0 — just-in, just-out). O sistema escolhe o bloco que está armazenado há mais tempo na cache, independentemente de estar sendo usado ou não com freqüência pela UCP.

O que tem menos referências (LFU — Least Frequently Used) — o sistema escolhe o bloco que tem tido menos acessos por parte da UCP (menos referências).

Escolha aleatória — trata-se de escolher aleatoriamente um bloco para ser substituído, independentemente de sua situação no conjunto.

Um estudo realizado sobre memórias cache, baseado em diversas simulações (ver SMIT82), obteve diversas conclusões, entre as quais a de que escolher um bloco aleatoriamente (não pode nem ser considerado um algoritmo) reduz muito pouco o desempenho do sistema em comparação com os demais algoritmos baseados em algum tipo de uso, e é extremamente simples de implementar.

5.5.4.4 Política de Escrita pela Memória Cache

Em sistemas com memória cache, toda vez que a UCP realiza uma operação de escrita, esta ocorre imediatamente na cache. Como a cache é apenas uma memória intermediária, não a principal, é necessário que, em algum momento, a MP seja atualizada, para que o sistema mantenha sua correção e integridade.

Antes que um bloco possa ser substituído na cache, é necessário considerar se ele foi ou não alterado na cache e se estas alterações também foram realizadas na MP. Caso contrário, isso significa que o bloco da cache está diferente do da MP e isso não pode acontecer, pois a MP precisa ser tão corretamente mantida quanto a cache.

Atualmente podem ser encontradas algumas políticas de escrita, cada uma contendo suas vantagens e desvantagens em relação às outras, no que se refere principalmente ao custo e ao desempenho.

O problema é complicado se levarmos em conta algumas ponderações, tais como:

- A memória principal pode ser acessada tanto pela cache quanto por elementos de Entrada e Saída (um dispositivo de acesso direto à memória, DMA, por exemplo). Neste caso, é possível que uma palavra da MP tenha sido alterada na cache e ainda não na MP e, assim, esta palavra da MP está desatualizada. Ou um elemento de E/S pode ter alterado a palavra da MP e, então, a palavra da cache é que estará desatualizada.
- A memória principal pode ser acessada por várias UCP, cada uma contendo sua memória cache. Neste caso, é possível que uma palavra da MP seja alterada para atender à alteração de uma cache específica de uma UCP, e as demais caches cujo conteúdo esteja ligado a esta palavra estarão desatualizadas.

Entre as técnicas conhecidas, temos:

- *Escrita em ambas* (*write through*)

  Nesta técnica, cada escrita em uma palavra da cache acarreta escrita igual na palavra correspondente da MP, assegurando validade permanente e igual ao conteúdo de ambas as memórias. Caso haja outros módulos UCP/cache, estes alterarão também suas caches correspondentemente.

- *Escrita somente no retorno* (*write back*)

  Esta técnica não realiza atualização simultânea, como a anterior, mas somente quando o bloco foi substituído e se ocorreu alteração. Em outras palavras, sempre que ocorrer uma alteração da palavra na cache, o quadro correspondente será marcado através de um bit adicional, que pode ser denominado ATUALIZA, por exemplo. Assim, quando o bloco armazenado no quadro específico foi substituído, o sistema verifica o valor do bit ATUALIZA; caso seja de valor igual a 1, então o bloco é escrito na MP, caso contrário, não.

- *Escrita uma vez* (*write once*)

  É uma técnica apropriada para sistemas multi UCP/cache, que compartilhem um mesmo barramento. Por ela, o controlador da cache escreve atualizando o bloco da MP sempre que o bloco correspondente na cache foi atualizado pela primeira vez. Este fato não só atualiza ao mesmo tempo ambos os blocos (como na técnica *write through*), mas também alerta os demais componentes que compartilham o barramento único. Estes são cientificados de que houve alteração daquela palavra específica e impedem seu uso. Outras alterações (escritas) naquele bloco apenas são realizadas, na cache local, pois o bloco somente é atualizado na MP quando foi substituído na cache.

Comparando-se as três políticas, podem ser estabelecidas algumas conclusões:

- Com a política *write through* pode haver uma grande quantidade de escritas desnecessárias na MP, com a natural redução de desempenho do sistema.
- A política *write back* minimiza aquela desvantagem, porém a MP fica potencialmente desatualizada para utilização por outros dispositivos a ela ligados, como o módulo de E/S, o que os obriga a acessar o dado através da cache, o que é um problema.
- A política *write once* pode ser conveniente, mas apenas para sistemas com múltiplas UCP, não sendo ainda muito usado.

O mesmo estudo sobre caches mencionado anteriormente [SMIT 82] mostra que a percentagem de escritas na memória é baixa, da ordem de 15%, o que aponta para uma simples política *write through*.

5.6 UM POUCO MAIS DE DETALHES

Este item procura acrescentar mais detalhes a alguns pontos relacionados com a tecnologia e funcionamento das memórias de semicondutores em sistemas de computação modernos. Trata-se de assuntos naturalmente mais avançados ou que se relacionam com programas de cursos de natureza específica, razão pela qual são abordados separadamente neste item.

5.6.1 Sobre Tecnologias de Fabricação

Conforme já exposto anteriormente neste capítulo, as memórias principal e cache de um sistema de computação moderno são constituídas de dispositivos de armazenamento fabricados com tecnologia de semicondutores, cuja principal característica é garantir o mesmo tempo de acesso (da ordem de poucos nanossegundos) a qualquer de suas células. Esta forma de atuar originou o nome Memória de Acesso Randômico ou Aleatório, ou simplesmente RAM; em contrapartida as antigas memórias de acesso seqüencial.

Também foi mostrado que, em termos de tecnologia de fabricação e processo de funcionamento, as memórias RAM evoluíram em diferentes tipos, a saber:

- memórias para Leitura/Escrita (L/E) ou R/W — que o mercado tradicionalmente chama de RAM e não L/E, como deveria;
- memórias somente para leitura ou ROM.

As memórias L/E, popularmente conhecidas como RAM (embora as memórias ROM também sejam RAM, porque o tempo de acesso a qualquer de suas células é igual), podem ser fabricadas através do uso de dois elementos diferentes, redundando em dois grandes tipos:

- a memória SRAM (RAM estática) e,
- a DRAM (RAM dinâmica).

As memórias do tipo SRAM são utilizadas na fabricação de memórias cache, L1 ou L2, enquanto as memórias DRAM são empregadas na constituição das memórias principais (MP), sendo, como já mencionado, conhecidas como RAM ou MP.

Assim, popularmente, usam-se os termos SRAM (para caches), DRAM ou simplesmente RAM (para memória principal) e ROM (parte da memória principal que não é volátil).

As memórias RAM (SRAM ou DRAM) são bem mais rápidas que as memórias ROM, razão por que em muitos sistemas se usa transferir o conteúdo da ROM BIOS para uma parte da RAM (MP) de modo a melhorar o desempenho da execução dos códigos da BIOS. Este processo de transferência para uso denomina-se *shadow*, sendo, no caso, *shadow the ROM BIOS*.

As memórias SRAM são constituídas exclusivamente de circuitos transistorizados, utilizando-se de tecnologia bipolar (a Fig. 5.28 apresenta o exemplo de diagrama de célula de memória SRAM com esta tecnologia) ou tecnologia NMOS — n-channel metal oxide semiconductor (ver Cap. 4), cujo diagrama de célula é mostrado na Fig. 5.29. Ambos são conhecidos como circuitos *flip-flop*, o qual retém o valor "setado" até que receba um sinal de *reset* ou perca energia (quando a alimentação é desligada).

Em outras palavras, memórias estáticas (as SRAM) são aquelas em que o valor de 1 bit permanece armazenado na célula enquanto houver energia elétrica alimentando o dispositivo. Isto é, se uma operação de escrita armazena, por exemplo, valor zero em uma célula, este valor permanece armazenado até que uma nova operação de escrita grave outro valor por cima (o valor 1, é claro). O valor, então, permanece estático.

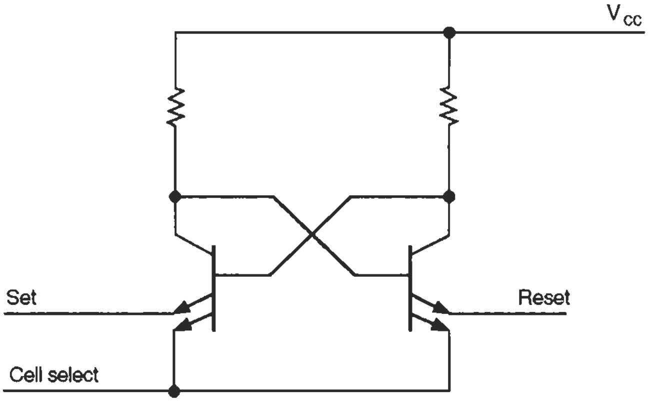


Figura 5.28 Célula de memória com tecnologia bipolar.

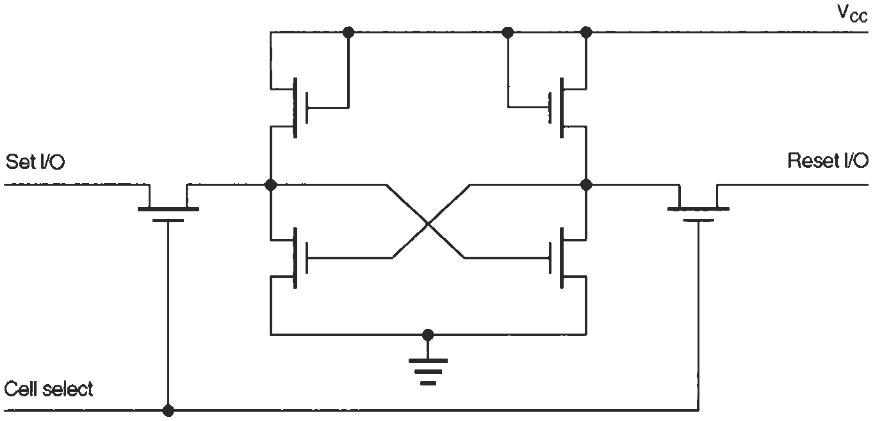


Figura 5.29 Célula estática (SRAM) utilizando tecnologia MOS.

As memórias DRAM (RAM dinâmicas) não armazenam o valor de 1 bit pelo mesmo processo das SRAM (circuitos transistorizados). O capacitor mostrado na Fig. 5.30 é a capacitância de entrada do transistor MOS. A carga armazenada neste capacitor (que permanece por alguns milissegundos) faz a célula guardar o valor 1, e a ausência de carga representa o valor 0. Elas representam, então, o valor de 1 bit através da presença ou ausência de carga elétrica do capacitor.[7] Este dispositivo trabalha de modo semelhante a uma bateria que, recebendo energia elétrica, armazena a carga por um tempo. Porém, com o passar do tempo ela vai perdendo a carga e, para mantê-la, precisa de uma recarga periódica, denominada em inglês *refresh*.

---

[7] Capacitor é um dispositivo eletrônico, como resistores e indutores, que recebendo uma carga elétrica mantém este valor armazenado por um certo tempo, perdendo aos poucos esta carga.

Na realidade, os capacitores mantêm sua carga por muito pouco tempo, sendo, portanto, necessário um recarregamento freqüente, milhares de vezes por segundo. Os poucos milissegundos de retenção do valor de carga logo passam, e o dado estaria perdido se não houvesse um processo de reescrita. Ou seja, periodicamente as células de uma memória DRAM requerem uma operação de reescrita dos dados, que se denomina *refreshing* (o termo "refrescar" não é ainda muito usado) e que é responsável pela velocidade menor das DRAM em relação às SRAM. Por isso, um ciclo de memória em uma DRAM é maior que o tempo de acesso.

Há vários métodos para realizar o recarregamento (*refresh*); em um deles o circuito de recarregamento lê o conteúdo de cada célula e a recarrega, antes que sua energia se desvaneça e impeça o circuito de "entender" o valor do bit que estava armazenado na referida célula. Normalmente, o recarregamento se realiza linha por linha. Uma outra técnica de recarregamento é denominada *self refresh*, ou seja, nela o próprio componente da memória realiza o recarregamento, independentemente da UCP ou de um circuito externo de recarregamento. Este método, construído no interior da pastilha DRAM, reduz bastante o consumo de energia, sendo, por isso, utilizado em notebooks.

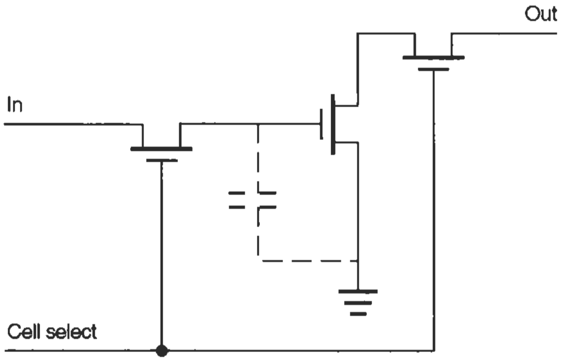


Figura 5.30 Célula de memória do tipo dinâmica (DRAM).

Efetuando-se uma breve comparação entre as memórias SRAM e DRAM podem-se identificar alguns pontos significativos de cada uma delas:

- As memórias SRAM não necessitam de recarga (*refresh*) para manter o valor de 1 bit armazenado, enquanto as memórias DRAM sim. Isso acontece devido ao processo de funcionamento delas e favorece a SRAM em termos de velocidade de acesso, pois o recarregamento das DRAM acarreta uma perda de tempo apreciável.
- As memórias DRAM ocupam menos espaço físico que as memórias SRAM, devido à menor quantidade de componentes requeridos para armazenar cada bit. A observação das Figs. 5.28 a 5.30 permite verificar esta afirmação: nas DRAM há um capacitor e apenas um transistor (responsável pela leitura do bit), enquanto nas SRAM há necessidade de vários transistores, o que acarreta mais espaço físico na pastilha. As DRAM ocupam cerca de ¼ da área das SRAM, o que se torna uma vantagem apreciável em termos de economia de espaço.
- As memórias SRAM carregam um custo maior de fabricação que as DRAM devido ao maior número de componentes (transistores) por bit que apresentam. Ainda que, nas DRAM, haja o acréscimo de custo para introdução do circuito de recarregamento, este acréscimo é bem menor que a economia de componentes, especialmente em memórias com grande capacidade de armazenamento.

5.6.1.1 Evolução da Tecnologia de Fabricação de Memórias DRAM

O processo de recarregamento (*refresh*) acarreta, sem dúvida, uma redução na velocidade de acesso das memórias DRAM, o que pode ser compensado pelo uso inteligente das memórias cache. No entanto, à medida que as velocidades dos processadores têm aumentado, nem o aumento das memórias cache (que ainda não pode ser demasiado devido ao custo) serve para compensar tanto esta baixa velocidade. Em razão disso, têm-se desenvolvido outras versões de memória DRAM com o propósito de reduzir o intervalo de velocidade UCP/MP.

É importante realçar, no entanto, que as diversas versões de DRAM que vêm surgindo no mercado não modificam substancialmente o problema da diferença de desempenho delas em relação à UCP. O aumento da velocidade e da capacidade das memórias cache, bem como o uso de mais de um nível de cache (L1/L2) tem produzido melhores resultados. Além disso, em grande parte das vezes parece mais importante para aumentar o desempenho do sistema ter-se MAIS quantidade de memória do que memórias ligeiramente mais rápidas. Não se deve esquecer, ainda, que, apesar da melhoria de velocidade e novas tecnologias de DRAM (que mostraremos em seguida), qualquer que seja a versão, uma DRAM é sempre a mesma, como no original.

Algumas dessas versões são apresentadas a seguir. A Tabela 5.2 apresenta um quadro descritivo dos diversos tipos de memória RAM.

DRAM original — é a memória com tecnologia DRAM mais antiga; utiliza o processo convencional de endereçamento linha/coluna (ver item 5.6.5 a seguir), isto é, primeiro é enviada a parte do endereço de linha e depois a parte da coluna. Apesar de antiga e de baixa velocidade, da ordem de 60 ns a 70 ns, ainda continua a ser usada em certos sistemas.

Fast Page Mode (FPM) DRAM — é uma tecnologia que produz memórias com velocidade pouco maior que as DRAM originais, tendo, no entanto, sido usada com freqüência em sistemas mais antigos, devido a sua excelente compatibilidade com diversos tipos de placa-mãe. Ela funciona de modo que, em vez de o controlador ter que enviar endereço de linha por linha, ele precisa apenas enviar o endereço da primeira linha de acesso e o endereço das colunas, de modo que o sistema vai lendo as células contíguas. Isso reduz o tempo de acesso em relação ao processo anterior.

Extended Data Out (EDO) DRAM — trata-se de um avanço em relação à tecnologia FPM, produzindo memórias com um pouco mais de velocidade. No caso, essas memórias possuem um circuito de tempo diferente, de modo que um acesso à memória possa ser iniciado **antes** que o anterior tenha terminado. Ela produz um aumento de desempenho da ordem de 5% sobre as memórias FPM e cerca de 20 a 25% sobre as DRAM originais. Este tipo tem sido bastante usado em sistemas Pentium e similares, porém com o aumento da velocidade dos processadores vem sendo substituída por versões de DRAM mais rápidas.

Burst Extended Data Out (BEDO) DRAM — é um tipo de tecnologia que produz resultados de desempenho melhores que os tipos anteriores, FPM e EDO. No caso, o sistema da memória acelera o processo de leitura através do envio de três endereços de coluna sucessivos, após a leitura de uma coluna, de maneira que 4 bits são lidos de uma vez (em modo *burst*). Apesar do incremento de velocidade, este tipo não é muito usado devido ao fato de os principais fabricantes de placas-mãe e processadores, especialmente a Intel, não terem manifestado interesse, preferindo o tipo SDRAM, apresentado a seguir. Na realidade, uma memória somente se torna popular se houver placas capazes de suportá-la.

Synchronous DRAM (SDRAM) — a grande diferença de tecnologia deste tipo para os anteriores consiste no fato de que as memórias DRAM anteriores funcionam de modo assíncrono, enquanto as SDRAM funcionam sincronizadas no relógio — velocidade (ver Cap. 6) do processador. A velocidade da memória SDRAM é informada em MHz (como a dos relógios) e não em nanossegundos, o que torna mais fácil a comparação entre a velocidade do barramento e a da memória embora alguns ainda usem os valores em ns, por exemplo, SDRAM de 83 MHz ou 12 ns e de 100 MHz ou 10 ns. A grande vantagem dessa tecnologia é o fato de usar a velocidade do processador e, por isso, poder funcionar com velocidades elevadas, o que as anteriores, assíncronas, não fazem, pois trabalham com velocidades menores, do barramento. Além disso, as SDRAM possuem outros aperfeiçoamentos em seu modo de funcionar, como o modo rajada (*burst*). Genericamente pode-se, também, mencionar duas categorias de memórias SDRAM, definidas em função da freqüência de trabalho do barramento: são chamadas PC-66 e PC-100. No primeiro tipo, trata-se de memórias que funcionam corretamente até uma freqüência máxima de 66 MHz, enquanto as PC-100 podem operar com os barramentos mais rápidos, de 100 MHz.

Direct Rambus DRAM (DRDRAM) — trata-se de uma nova tecnologia de fabricação de memórias RAM bastante diferente das anteriores, sendo propriedade de um fabricante denominado Rambus Inc., não sendo, pois, de uso público. No entanto, a Intel, como a AMD e Cyrix, vem se posicionando altamente favoráveis ao seu uso, o que torna a RDRAM um tipo freqüente nos futuros microcomputadores, como as SDRAM. Esta memória usa um barramento próprio, desenvolvido pela empresa, que lhe permite altíssimas velocidades de transferência de dados, da ordem de 800 MHz. Entretanto, ele transfere 16 bits de cada vez em vez de 64 como nas SDRAM, o que não é problema em razão das elevadas taxas de transferência, que suprem a diferença de largura de bits do barramento.

Observação:

É importante ressaltar uma notável diferença de características entre esses incontáveis tipos de tecnologia de fabricação de memórias RAM, que é o fato de algumas serem assíncronas e outras síncronas. Uma memória trabalha no modo assíncrono (as de tipos mais antigos, operando com barramentos e processadores de menor velocidade), quando seu funcionamento não é cadenciado, não é regido pelo batimento do relógio do processador (temos que mencionar neste ponto um elemento de informação — relógio — que será detalhado apenas no próximo capítulo, mas que é crucial para nossa explicação), enquanto uma memória do tipo síncrona funciona na freqüência do sistema, o que lhes permite naturalmente maiores velocidades de acesso. As memórias do tipo DRAM original, FPM, EDO e BEDO são assíncronas, e as SDRAM e RDRAM são do tipo síncronas.

5.6.1.2 Sobre a Apresentação dos Elementos de Memória no Sistema de Computação

A quantidade de informações e de nomenclatura de um mesmo assunto na Informática (vide o presente assunto de memórias) é considerável, o que, na maioria das vezes, pode confundir o leitor menos comprometido com a tecnologia em questão. É caso, por exemplo, das nomenclaturas EDO DRAM, SDRAM, SIMM etc. Trata-se da mesma coisa? Será SIMM mais uma tecnologia de fabricação? Como discernir ou escolher memórias na ocasião de adquirir um sistema?

Bem, os termos DRAM e SIMM significam coisas diferentes relacionadas com fabricação e instalação das memórias em um computador. Um item ou grupo de itens (DRAM, SRAM, EDO DRAM, RDRAM etc.) refere-se ao tipo de tecnologia de fabricação de uma memória, ao modo pelo qual os circuitos físicos são dispostos na pastilha e ao seu processo de funcionamento para realizar operações de leitura e de escrita. Já os termos SIMM, DIMM etc. estão relacionados com o modo pelo qual as pastilhas são instaladas na placa de circuito impresso dos computadores.

Os modelos atuais de módulos de memória denominam-se SIMM (Single In Line Memory Module), que podem ser do tipo SIMM-30 e SIMM-72, DIMM (Double In Line Memory Module) e RIMM (Rambus In Line Memory Module), cujos formatos são apresentados, respectivamente, nas Figs. 5.31, 5.32, 5.33 e 5.34.

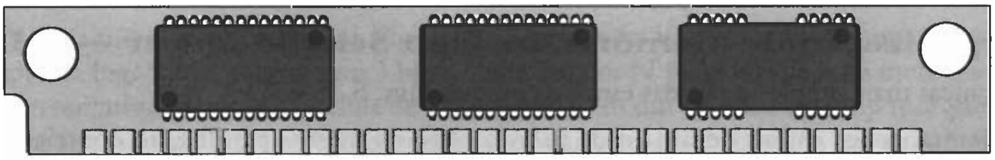


Figura 5.31 Módulo SIMM-30.

O módulo SIMM-30 possui 30 terminais (endereços, dados e controle) e permite a passagem de 8 bits em cada ciclo do barramento, enquanto o módulo SIMM-72 possui 72 terminais, permitindo a transferência de 32 bits em cada ciclo. Se estes módulos permitirem o emprego de método de detecção de erros (bit de paridade), então serão capazes de transferir 9 e 36 bits, respectivamente, um a mais para cada grupo de 8 bits. O módulo DIMM possui 168 terminais, possibilitando a transferência de 64 bits de cada vez, sendo atualmente empregado com memórias SDRAM; finalmente, o módulo RIMM, propriedade da empresa Rambus Inc., é utilizado apenas para receber pastilhas de memória RDRAM.

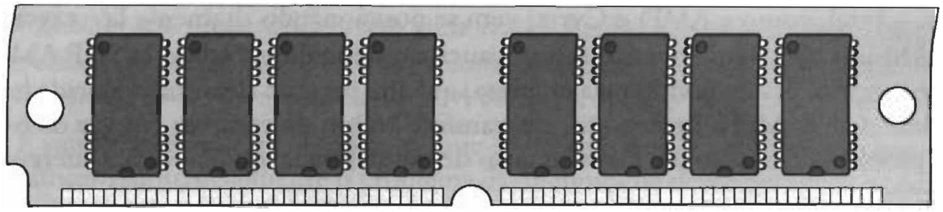


Figura 5.32 Módulo SIMM-72.

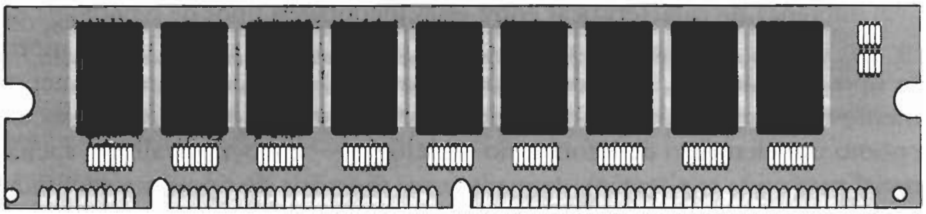


Figura 5.33 Módulo DIMM.

Como exposto, os elementos SIMM, DIMM e RIMM são placas utilizadas para facilitar a instalação das pastilhas de memória, o que se faz apenas por encaixe nos circuitos impressos, em vez do processo antigo de soldar os pinos da própria pastilha, com todos os óbvios inconvenientes deste processo. Nas referidas placas podem ser fixadas pastilhas de qualquer tipo de tecnologia, como DRAM, EDO DRAM etc.

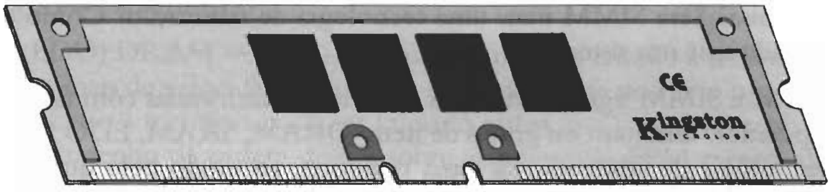


Figura 5.34 Módulo RIMM.

5.6.2 Localização da Célula Desejada em uma Operação de Leitura ou de Escrita

Qualquer que seja a tecnologia de fabricação de MP, o processo de localização de uma determinada célula para efeito de uma operação de leitura ou escrita é o mesmo, seja qual for, já que existe mais de um método para isso.

5.6.2.1 Organização de Memória do Tipo Seleção Linear — 1 Dimensão

Uma das técnicas mais simples e rápidas está descrita nas Figs. 5.35 e 5.36.

Com esta técnica, todos os bits de uma dada palavra estão na mesma pastilha, ao contrário de outra (por matriz de linha/coluna), em que os bits de cada palavra estão espalhados por várias pastilhas (por exemplo, um sistema com 16 pastilhas superpostas contém em cada uma 1 bit de uma palavra de 16 bits).

No primeiro caso, além de todos os bits de uma palavra estarem na mesma célula, o arranjo físico das células é o mesmo arranjo lógico das palavras na memória. O conjunto é organizado em N palavras de M bits cada (tal como descrito no item 5.3.1). Por exemplo, uma pastilha de 16K bits pode conter 1024 palavras de 16 bits cada. Os elementos de cada conjunto são conectados por linhas horizontais (filas) e verticais (colunas). Cada linha horizontal da memória (uma célula) é uma saída do decodificador de linha (*select*), e cada linha vertical da memória (1 bit da palavra) se conecta ao sensor de dados para receber ou enviar 1 bit da palavra.

Em outras palavras, o decodificador tem 2^E saídas para E entradas, isto é, se cada endereço armazenado no REM (MAR) tem E bits, a ligação REM-decodificador tem E linhas (ver Fig. 5.35). A saída do decodificador

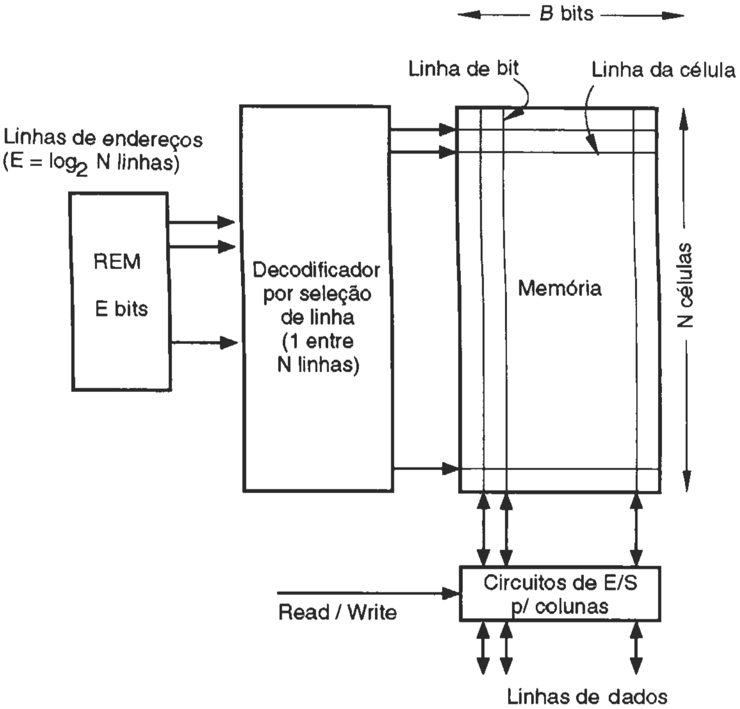


Figura 5.35 Organização de memória do tipo seleção linear — 1 dimensão.

terá 2^E linhas, uma para cada um dos $N = 2^E$ célula da memória. No exemplo, a pastilha de 16K bits teria um REM de 10 bits, 10 linhas de saída do REM para o decodificador e deste sairiam 1024 linhas, uma para cada palavra (célula) da memória.

Tomemos, por exemplo, o endereço 12 decimal ou 000000001100 em binário, armazenado no REM. Isto acarretaria uma saída 1 na 13.ª linha do decodificador, correspondente ao endereço 12 (13.ª linha, porque o 1.º endereço é 0). As demais linhas seriam iguais a zero.

Vamos tentar entender melhor através de um exemplo simples, de uma MP com 12 bits distribuídos em quatro células de 3 bits cada.

A Fig. 5.36(a) mostra um esquema de uma única célula, a Fig. 5.36(b) mostra o esquema mais completo da memória, com endereço de 2 bits (porque são 4 células e $2^2 = 4$) e célula com palavra de 3 bits, e a Fig. 5.36(c) mostra o decodificador do mesmo exemplo, mas de forma mais expandida.

A célula básica da memória mostrada na Fig. 5.36(a) representa um único bit de uma palavra qualquer da MP (no exemplo da Fig. 5.36 a palavra tem 3 bits). Cada um dos N × M bits de uma memória qualquer teria um circuito representativo idêntico. A célula de 1 bit consiste em um circuito flip-flop (porque é uma memória SRAM), com seu circuito de controle associado. Ao usar uma célula baseada em circuito flip-flop, o sistema de controle da memória deve adotar uma técnica para selecionar qual a célula desejada, bem como um método para controlar se a operação sobre a célula escolhida será de escrita ou de leitura.

A Fig. 5.36(b) mostra um esquema mais completo de toda a MP. Dado um endereço, o decodificador ativa a linha da célula correspondente (ativar significa gerar um bit 1 na saída do decodificador).

O REM (MAR) possui capacidade para armazenar 2 bits (cada endereço desta memória tem 2 bits) e, portanto, ele possui 2 flip-flops, um para cada bit. O REM está ligado ao decodificador, que possui quatro diferentes saídas, uma para cada valor de endereço (00, 01, 10 e 11), da mesma forma que o exemplo genérico da Fig. 5.35. Cada uma das 4 saídas do decodificador está ligada a uma célula (palavra) da memória que, por sua vez, possui 3 flip-flops, um para cada bit da palavra. Cada flip-flop mostrado em bloco fechado na Fig. 5.36(b) possui os mesmos elementos do esquema de 1 flip-flop, apresentado nas Figs. 5.28, 5.29 e 5.36(a).

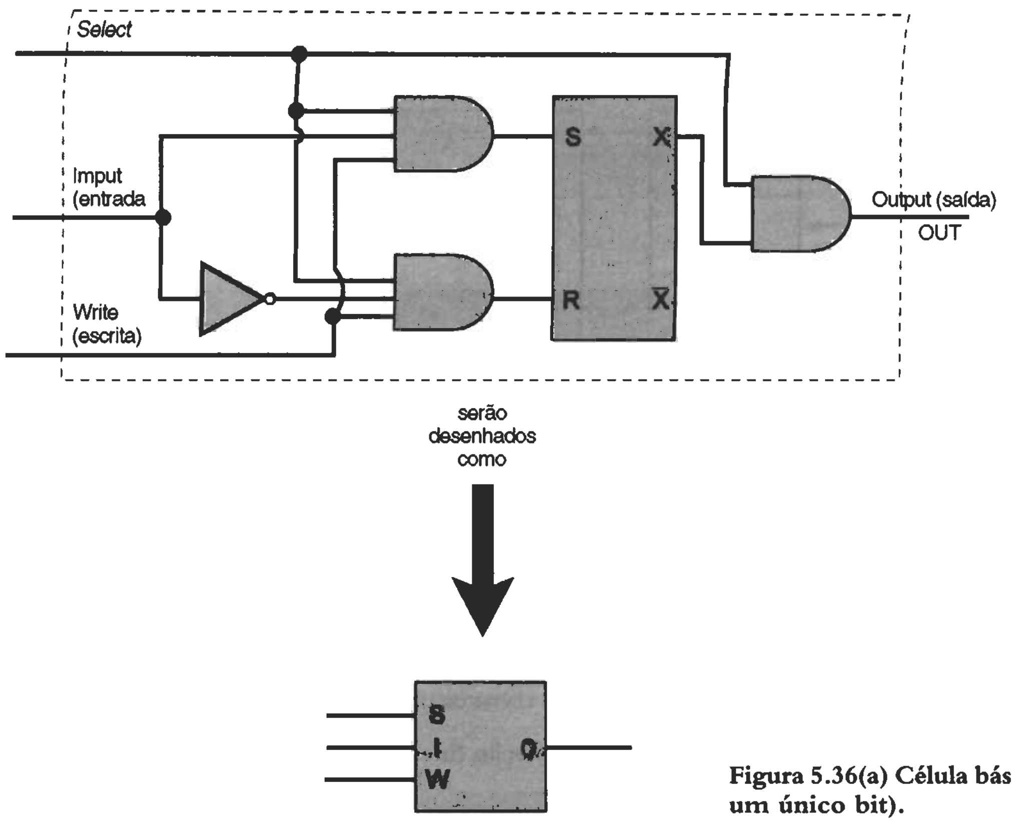


Figura 5.36(a) Célula básica de memória (representa um único bit).

A Fig. 5.36(c) mostra uma visão expandida do decodificador, com suas duas entradas, circuitos lógicos e quatro saídas. Por exemplo, se o REM contiver um valor 0 em cada um de seus flip-flops, então a linha superior do decodificador (correspondente ao endereço 00) será ativada, isto é, aparecerá o valor 1; as linhas de saída restantes permanecerão com 0. Da mesma forma, se o REM contiver 0 no 1.º flip-flop e 1 no 2.º flip-flop (endereço 01), a 2.ª linha de cima para baixo será ativada (bit 1) e as demais ficarão com o valor 0. Para cada entrada possível, uma única linha de saída do decodificador será selecionada, assumindo o valor 1 (ver a Fig. 5.36(a)).

Quando o REM seleciona uma célula da memória, pela passagem do endereço nele armazenado para o decodificador, e a linha READ (leitura) está ativada (bit 1), então o conteúdo dos 3 flip-flops correspondentes àquele endereço será transferido através das linhas O_1, O_2 e O_3. Se a linha WRITE (escrita) estiver com o valor 1 (ativada), os valores em I_1, I_2 e I_3 serão lidos para a memória.

A porta lógica AND conectada às linhas OUT (saída) das células de memória da Fig. 5.36 deve ter uma propriedade especial que faz a saída geral ser de valor 1 (verdadeiro) se qualquer uma das linhas tiver valor 1 (se qualquer OUT for igual a 1, então a linha por onde passam todos os OUT assume valor 1; caso contrário, ela assume valor 0). No item 4.4.3.2 vimos se chamar circuito *wired-OR*. Na Fig. 5.36(b), todas as células da primeira coluna são 11 *wired-OR*, de modo que, se uma das linhas de saída tomar o valor 1, a linha inteira terá o valor 1 (é a linha que chega à parte lógica O_1 ou O_2 ou O_3). Esta propriedade tem a maior importância e, de um modo geral, os circuitos de memória de semicondutores funcionam desta forma.

Para concluir, vamos ver o que acontece quando ocorre uma operação de leitura (a linha READ assume o valor 1) ou uma operação de escrita (a linha WRITE assume o valor 1).

Se a linha READ (ver Fig. 5.36(b)) assumir o valor 1, os valores de saída para os 3 flip-flops da linha (célula) selecionada serão apresentados na linha de saída de cada bit da célula (são 3 linhas de saída, uma O_1, outra O_2 e a última O_3). Por exemplo, se a 1.ª linha na memória tiver armazenado o valor 101 e se o REM tiver o valor 10 armazenado, a 3.ª linha de saída do decodificador (marcada 10) assumirá o valor 1, e as portas de entrada e de saída dos 3 elementos da célula serão selecionadas, restando saber se a operação é de leitura ou de escrita.

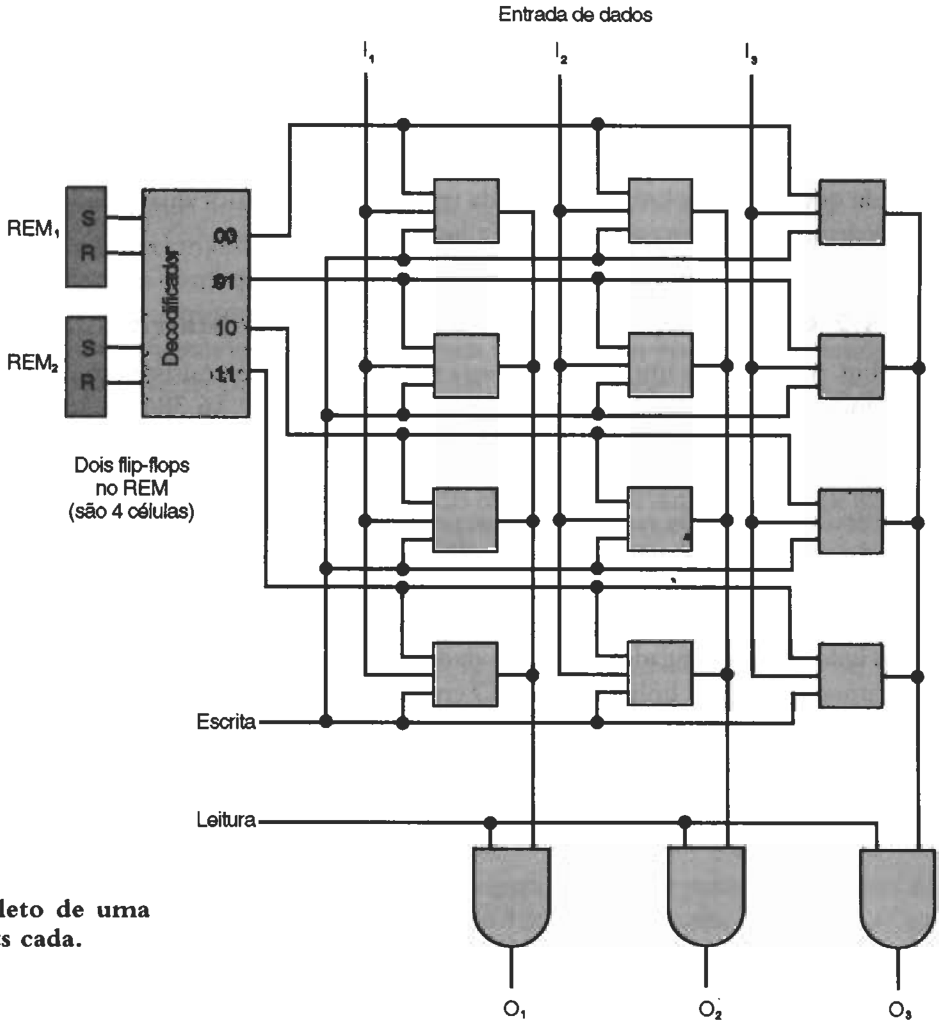

Entrada de dados
I₁
I₂
I₃
REM₁
S
R
REM₂
S
R
Decodificador
00
01
10
11
Dois flip-flops
no REM
(são 4 células)
Escrita
Leitura
O₁
O₂
O₃

Figura 5.36(b) Esquema completo de uma memória com 4 células de 3 bits cada.

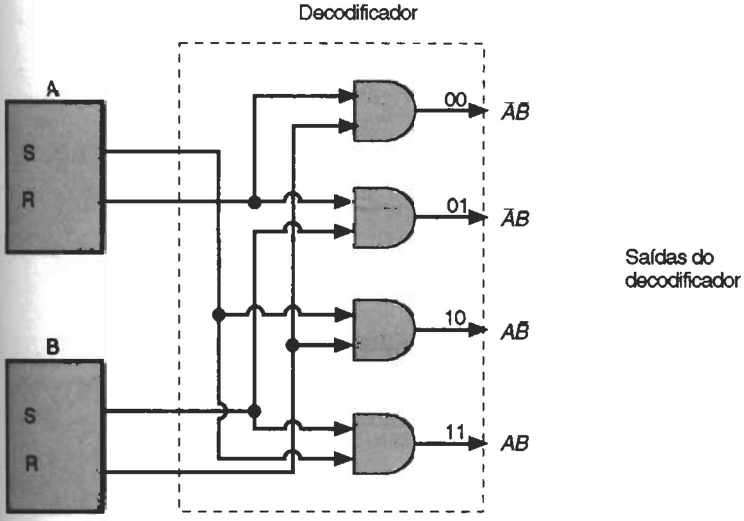

Decodificador
A
S
R
B
S
R
00
01
10
11
ĀB̄
ĀB
AB̄
AB
Saídas do
decodificador

Figura 5.36(c) Visão expandida do decodificador, com 2 entradas e 4 saídas.

Quando a linha READ se tornar 1 (a operação é de leitura), a saída dos 3 elementos da 3.ª célula (de endereço 10) será enviada para as portas AND O_1, O_2 e O_3 (abaixo da figura), as quais transmitirão o valor 101 para o destino.

Se a linha WRITE se tornar 1 (a operação é de escrita) e o REM novamente tiver o valor 10, os valores de entrada em I_1, I_2 e I_3 serão transferidos para os flip-flops da terceira fila.

Esta memória, exemplificada com a descrição e a Fig. 5.36(b), é completa, sendo capaz de realizar operações de escrita e de leitura. Porém, à medida que a quantidade de palavras (células) aumenta, o decodificador tem que crescer demasiadamente e os circuitos se tornam mais complexos, apesar da grande vantagem de velocidade que ela tem.

Uma alternativa para reduzir a complexidade dos circuitos e o tamanho do decodificador, que crescem à medida que haja mais linhas de entrada (memória com maior quantidade de células), é utilizar uma organização denominada *organização por matriz linha/coluna.*

5.6.5.2 Organização de Memória do Tipo Matriz Linha/Coluna

A Fig. 5.37 mostra um exemplo dessa organização, que consiste em uma memória com 16K células, onde cada endereço é um número com 14 bits, visto que 2^{14} = 16.384 ou 16K. O diagrama mostrado na figura é o mesmo, seja a memória estática (SRAM), seja dinâmica (DRAM), com tecnologia bipolar ou MOS.

Cada pastilha (*chip*) possui 14 linhas de endereço, identificadas pela nomenclatura A_0 a A_{13}. Essas linhas de endereço são divididas em duas partes: uma delas, compreendendo as 7 primeiras linhas, de A_0 a A_6, são conectadas ao *decodificador de colunas;* a outra compreende as linhas de A_7 a A_{13}, que são conectadas ao *decodificador de linha.* Assim, o decodificador de colunas deve decodificar um endereço de 7 bits (e não de 14 bits — endereço completo), produzindo a ativação de uma das 128 linhas (2^7 = 128) e não de 16.384 linhas (2^{14}). Por outro lado, o decodificador de linhas decodificará o endereço de 7 bits que lhe corresponde, ativando, também, uma de suas 128 linhas de saída. O cruzamento da linha e coluna ativadas pelos respectivos decodificadores corresponde à célula desejada (endereçada pelos 14 bits). Verifica-se, nesse exemplo, que ocorrem 128 × 128 = 16.384 cruzamentos, cada um correspondendo a uma das 16.384 = 16K células da memória.

Uma vez selecionada a respectiva célula, o sistema, então, poderá efetuar uma leitura do conteúdo da célula ou uma escrita, gravando um valor na célula em questão.

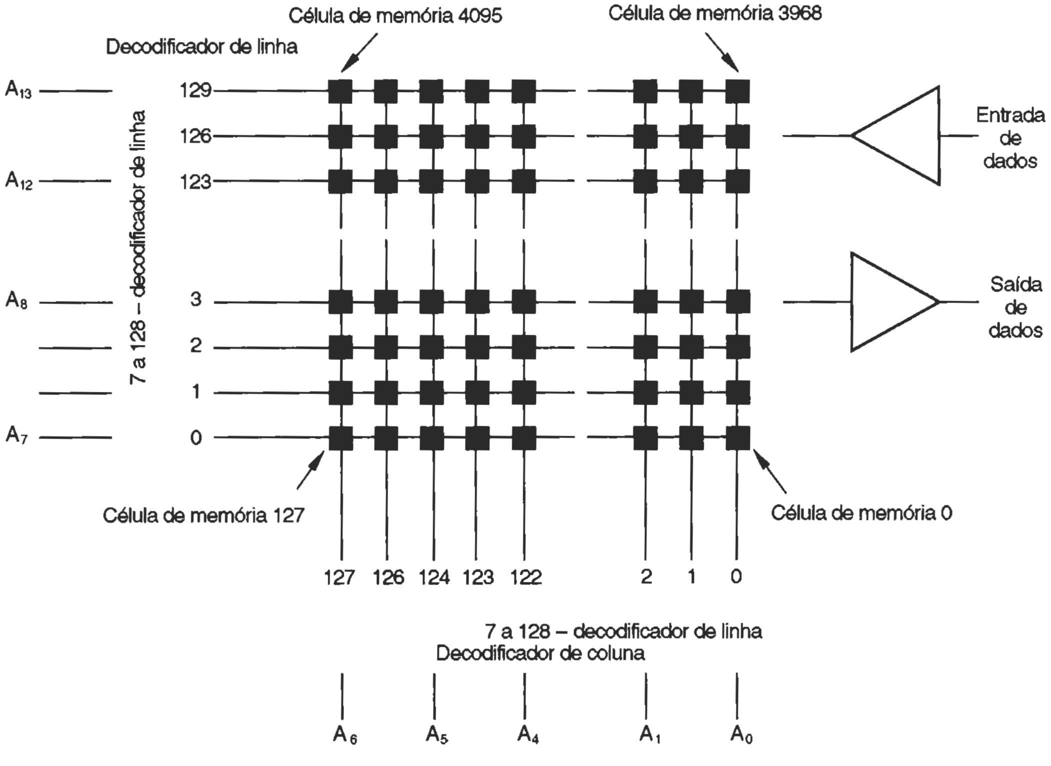

Célula de memória 4095
Célula de memória 3968
Decodificador de linha
A_{13}
A_{12}
A_8
A_7
129
126
123
3
2
1
0
7 a 128 – decodificador de linha
Entrada de dados
Saída de dados
Célula de memória 127
Célula de memória 0
127
126
124
123
122
2
1
0
7 a 128 – decodificador de linha
Decodificador de coluna
A_6
A_5
A_4
A_1
A_0

Figura 5.37 Organização de memória por matriz de linha/coluna.

Uma breve comparação entre as duas técnicas encontra algumas desvantagens na primeira delas, a de *seleção linear*, embora com uma boa vantagem relativa ao baixo tempo de acesso, quando se tratar de poucas células.

Em primeiro lugar, a quantidade de linhas de saída do decodificador é muito maior no esquema de seleção linear (uma para cada endereço) do que no de matriz por linha e coluna (a quantidade de linhas de saída em cada um dos dois decodificadores é igual à raiz quadrada do total de endereços da memória — no exemplo mostrado era $\sqrt{16.384} = 128$).

Há uma maior quantidade de lógica (mais circuitos) a ser inserida na pastilha de uma memória que emprega a técnica de seleção linear do que na de uma outra que usa matriz por linhas e colunas. Isso porque a matriz por linhas/colunas é um conjunto mais ou menos quadrado, enquanto a seleção linear produz um conjunto retangular estreito e comprido, consistindo em um grande número de palavras (células) com poucos bits em cada uma (por isso estreito). Cada linha de palavra deve ter um acionador do sinal elétrico (*signal driver*) e cada um deles deve ser conectado a uma porta lógica do decodificador; para cada bit ainda é necessário um amplificador.

Enquanto isso, em uma técnica de matriz por linhas e colunas que seleciona 1 bit da palavra por pastilha (e várias pastilhas para totalizar a quantidade de bits de uma palavra), são usados menos amplificadores e acionadores de sinal (duas vezes a raiz quadrada das células).

Consideremos um bom exemplo decorrente desta comparação, descrito em [STAL87]:

Uma pastilha de 256K bits pode ser usada, por exemplo, para uma memória com organização de seleção linear, para armazenar 8K palavras de 32 bits cada uma. Isto acarreta um total de circuitos (simplificadamente) de 16K.

Utilizando-se a organização de matriz de linhas × colunas, obtém-se a raiz quadrada de 256K, que é 512, o que acarreta um total de circuitos da ordem de 1K, uma considerável vantagem.

EXERCÍCIOS

1) Um computador possui uma memória principal com capacidade para armazenar palavras de 16 bits em cada uma de suas N células e o seu barramento de endereços tem 12 bits de tamanho. Sabendo-se que em cada célula pode-se armazenar o valor exato de uma palavra, quantos bytes poderão ser armazenados nessa memória?

2) O que você entende por acesso à memória? Caracterize o tempo de acesso nos diversos tipos de memória.

3) Quais são as possíveis operações que podem ser realizadas em uma memória?

4) Qual é a diferença conceitual entre uma memória do tipo SRAM e outra do tipo DRAM? Cite vantagens e desvantagens de cada uma.

5) Qual é a diferença, em termos de endereço, conteúdo e total de bits, entre as seguintes organizações de MP:

   a) memória A: 32K células de 8 bits cada;

   b) memória B: 16K células de 16 bits cada; e

   c) memória C: 16K células de 8 bits cada.

6) Qual é a função do registrador de endereços de memória (REM)? E do registrador de dados de memória (RDM)?

7) Descreva os barramentos que interligam UCP e MP, indicando função e direção do fluxo de sinais de cada um.

8) Descreva passo a passo uma operação de leitura. Utilize um diagrama esquemático.

9) Faça o mesmo para uma operação de escrita.

10) Um computador possui um RDM com 16 bits de tamanho e um REM com capacidade para armazenar números com 20 bits. Sabe-se que a célula deste computador armazena dados com 8 bits de tamanho e que ele possui uma quantidade N de células, igual à sua capacidade máxima de armazenamento. Pergunta-se:

a) Qual é o tamanho do barramento de endereços?

b) Quantas células de memória são lidas em uma única operação de leitura?

c) Quantos bits têm a memória principal?

11) Um microcomputador possui uma capacidade máxima de memória principal (RAM) com 32K células, cada uma capaz de armazenar uma palavra de 8 bits. Pergunta-se:

a) Qual é o maior endereço, em decimal, desta memória?

b) Qual é o tamanho do barramento de endereços deste sistema?

c) Quantos bits podem ser armazenados no RDM e no REM?

d) Qual é o total máximo de bits que pode existir nesta memória?

12) Considere uma célula de uma MP cujo endereço é, em hexadecimal, 2C81 e que tem armazenado em seu conteúdo um valor igual a, em hexadecimal, F5A. Sabe-se que, nesse sistema, as células têm o mesmo tamanho das palavras e que em cada acesso é lido o valor de uma célula. Pergunta-se:

a) Qual deve ser o tamanho do REM e do RDM nesse sistema?

b) Qual deve ser a máxima quantidade de bits que pode ser implementada nessa memória?

13) Considere uma memória com capacidade de armazenamento de 64 Kbytes. Cada célula pode armazenar 1 byte de informação e cada caractere é codificado com 8 bits. Resolveu-se armazenar na memória deste sistema um conjunto de caracteres do seguinte modo: a partir do endereço (hexadecimal) 27FA, foram escritos sucessivamente grupos de 128 caracteres iguais, iniciando pelo grupo de As, seguido do grupo de Bs, e assim por diante. Qual deverá ser o endereço correspondente ao local onde está armazenado o 1.° J?

14) Sugira razões pelas quais as RAM são freqüentemente organizadas com somente 1 bit por pastilha (organização de matriz/coluna) e ROM são usualmente organizadas com a palavra inteira em uma pastilha (organização do tipo linear).

15) Uma memória cache associativa por conjunto consiste em 64 quadros divididos em conjuntos de 4 quadros cada. A memória principal contém 4K blocos de 128 palavras cada um. Mostre o formato de um endereço de MP.

16) Determine a quantidade de portas AND que deverá ser colocada na saída de uma memória de 4096 células de 1 bit cada uma, cuja organização é do tipo linear.

17) O custo das memórias SRAM é maior que o das memórias DRAM. No entanto, o processo de conexão das memórias DRAM é mais complexo que o das SRAM e, em conseqüência, o preço do interface das DRAM é bem maior que o das SRAM. Supondo que um interface de DRAM custe R$5,00, um interface de SRAM custe R$1,00, que o preço por bit de uma SRAM seja de R$0,00002 e o de uma DRAM de R$0,00001, calcule quantos bits deve ter uma memória dinâmica (DRAM) para que o conjunto seja mais barato.

18) Calcule quantos flip-flops devem existir em um REM e em um RDM de um sistema de computação cuja memória seja constituída de um conjunto de 1M células de 8 bits cada uma.

19) Qual é a diferença funcional entre uma memória do tipo assíncrona e uma do tipo síncrona?

20) Compare uma memória principal e uma memória cache em termos de tempo de acesso, capacidade e temporariedade de armazenamento de dados.

21) Em que circunstâncias uma cache que funciona com mapeamento associativo por conjunto pode ser considerada igual à cache que funciona com mapeamento direto?

22) No que se refere a memórias de um sistema de computação o que significa genericamente o termo SIMM? E o termo EDO DRAM? Eles têm semelhança entre si?

23) Uma memória ROM pode ser também considerada uma memória do tipo Leitura/Escrita? Por quê?

24) Considere um sistema de computação que possui uma memória principal (RAM) com capacidade máxima de endereçamento de 64K células, sendo que cada célula armazena um byte de informação. Para criar um sistema de controle e funcionamento da sua memória cache, a memória RAM é constituída de blocos de 8 bytes cada. A memória cache do sistema é do tipo mapeamento direto, contendo 32 linhas. Pergunta-se:

    a) Como seria organizado o endereço da MP (RAM) em termos de etiqueta (tag), número da linha e do byte dentro de uma linha?

    b) Em que linha estaria contido o byte armazenado no seguinte endereço da MP: 0001 0001 0001 1011?

    c) Qual é a capacidade da memória cache em bytes?

25) Qual é a diferença entre uma memória do tipo PROM e uma do tipo EPROM?

26) E qual é a diferença entre uma memória do tipo ROM "pura" (original) e uma memória do tipo PROM? E o que é idêntico nelas?

27) O que significa o termo *shadow ROM*?

28) Enumere os diferentes tipos de memória que podem existir em um microcomputador moderno, atual, desde um simples registrador até os CD-ROM etc.

6

Unidade Central de Processamento

6.1 INTRODUÇÃO

No Cap. 2 apresentamos uma visão global da estrutura sistêmica de um computador e uma breve descrição de cada um de seus principais componentes. A partir do Cap. 5 iniciamos a descrição mais detalhada de cada componente, a memória em primeiro lugar, mais especificamente a memória principal e a cache, ambas interligadas direta e logicamente à UCP na execução de um programa.

Neste capítulo, analisaremos um outro componente, o *processador central* ou Unidade Central de Processamento — UCP, que desempenha um papel crucial no funcionamento do sistema de computação. O processador é responsável pela atividade-fim do sistema, isto é, computar, calcular, processar. Como já mencionamos em capítulos anteriores, os processadores atuais são fabricados de modo que, em um único invólucro (pastilha — *chip*), são inseridos todos os elementos necessários à realização de suas funções. Cada vez mais, a tecnologia tem avançado nessa área, de modo a se fabricar processadores mais complexos e poderosos, como o Pentium III, da Intel, cuja pastilha é fabricada contendo cerca de 10 milhões de transistores, ou o Pentium III Xeon, com cerca de 28 milhões de transistores, ou ainda o Athlon (K7), da AMD, que é encapsulado com cerca de 22 milhões de transistores. A Tabela 6.1 apresenta um quadro demonstrativo com as principais características de alguns processadores, o que nos permitirá comparar e observar sua evolução tecnológica.

Para apresentar os processadores, em primeiro lugar serão apresentadas suas funções básicas, qualquer que seja seu modelo, descrevendo-se os dispositivos essenciais que realizam as referidas funções. Em seguida, será descrito o funcionamento, passo a passo, deste processador básico. Finalmente, a exemplo do que foi incluído no capítulo sobre a memória, há um item destinado aos que desejam um pouco mais de detalhes sobre o processador e seus elementos internos. Neste item, trataremos com um pouco mais de profundidade os seguintes tópicos: arquitetura de processadores reais; a unidade aritmética e lógica, UAL; a arquitetura do tipo linha de montagem ou *pipelining*; e ainda barramentos: como funcionam e quais os principais tipos atuais; tipos de controle de processamento: programado no hardware e microprogramado.

6.2 FUNÇÕES BÁSICAS DA UCP

O *processador central* ou UCP é o componente vital do sistema de computação. Na realidade, a UCP é responsável pela realização de qualquer operação realizada por um computador. Isto quer dizer que a UCP comanda não somente as ações efetuadas internamente, como também, em decorrência da interpretação de uma determinada instrução, ela emite os sinais de controle para os demais componentes do computador agirem e realizarem alguma tarefa.

As Figs. 6.1(a), (b), (c) e (d) mostram exemplos de processadores, existentes em uma única pastilha, como o Intel 80486, Intel Pentium I (original), o AMD Athlon (K7) e Cyrix x86.

Um processador tem, por propósito, realizar operações com os dados (chamadas de processamento) normalmente numéricos. Para realizar essas operações, o processador necessita, em primeiro lugar, interpretar que tipo de operação é a que ele irá executar (pode ser a soma de dois números, pode ser a subtração de dois valores e assim por diante). Em seguida, antes da realização propriamente dita da operação é necessário que os dados estejam armazenados no dispositivo que irá executar a operação.

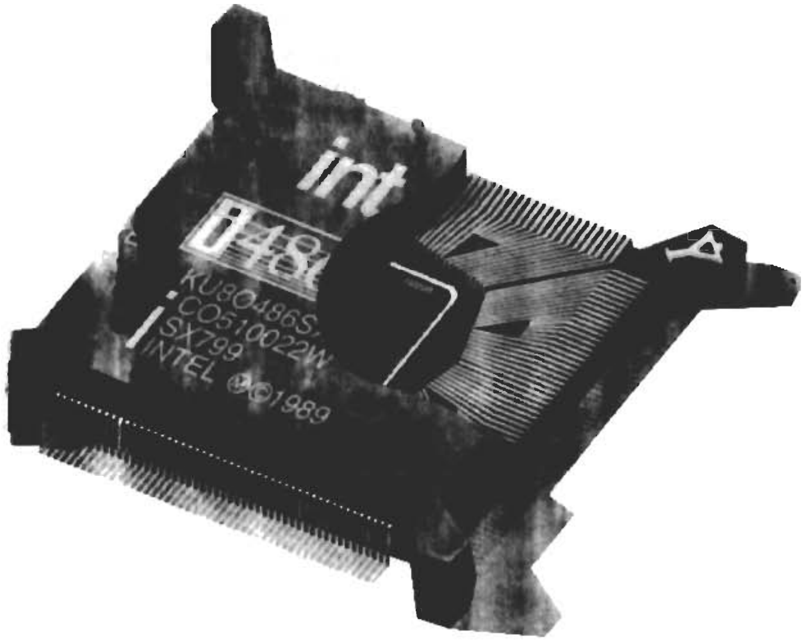


Figura 6.1(a) O processador Intel 80486.

Portanto, a UCP não somente realiza o processamento (executa a operação com os dados), como também controla todo o funcionamento do sistema (busca a descrição da operação a ser realizada — chamada instrução; interpreta que tipo de operação deverá ser realizado; localiza e busca os dados que serão processados e assim por diante).

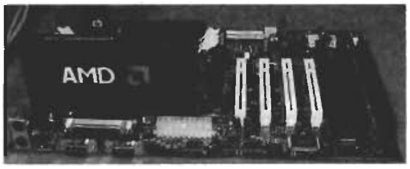


Figura 6.1(b) O processador AMD Athlon (k7)

intel.
pentium
150 MHz
intel.
pentium
166 MHz

Figura 6.1(c) O processador Cyrix x86.

Figura 6.1(d) O processador Intel Pentium I.

Todo processador é construído de modo a ser capaz de realizar algumas operações, denominadas primitivas, tais como: somar, subtrair, mover um dado de um local de armazenamento para outro, transferir um valor (dado) para um dispositivo de saída, etc. Essas operações e a localização dos dados que elas manipulam têm que estar representadas na única forma inteligível pelo sistema, que é uma seqüência de sinais elétricos, cuja intensidade corresponde a 0s e 1s (uma seqüência de bits).

A seqüência de 0s e 1s que formaliza uma determinada operação a ser realizada pelo processador denomina-se *instrução de máquina.*

Uma instrução de máquina é a identificação formal do tipo de operação a ser realizado (portanto, cada operação consiste em uma instrução diferente), contendo um grupo de bits que identifica a operação a ser realizada e outro grupo de bits que permite a localização e acesso aos dados que serão manipulados na referida operação. Ou seja, se a operação desejada é uma soma, a instrução de máquina correspondente deve conter os bits necessários para indicar que se trata de soma e onde estão armazenados os valores que deverão ser somados.

Um programa executável (ver Cap. 9) é constituído de um conjunto de instruções de máquina (ver o item 6.3) seqüencialmente organizadas. Para que a execução do referido programa tenha início é necessário que:

1) as instruções a serem executadas estejam armazenadas em células sucessivas, na memória principal;
2) o endereço da primeira instrução do programa esteja armazenado na UCP para que o processador possa buscar essa primeira instrução (veremos mais adiante que os endereços das instruções são armazenados em um registrador específico para isso).

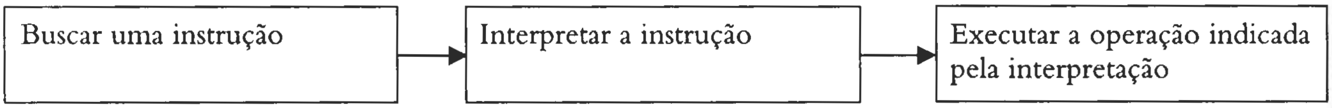
A função do processador central (UCP) consiste, então, em:

a) buscar uma instrução na memória (operação de leitura), uma de cada vez (cujo endereço deve estar armazenado no registrador existente na UCP e específico para esse fim);
b) interpretar que operação a instrução está explicitando (pode ser uma soma de dois números, uma multiplicação, uma operação de entrada ou de saída de dados, ou ainda uma operação de movimentação de um dado de uma célula para outra);
c) buscar os dados onde estiverem armazenados, para trazê-los até a UCP;
d) executar efetivamente a operação com o(s) dado(s), guardar o resultado (se houver algum) no local definido na instrução; e, finalmente,
e) reiniciar o processo buscando uma nova instrução.

Estas etapas compõem o que se denomina um *ciclo de instrução.* Este ciclo se repete indefinidamente (ver Fig. 6.2) até que o sistema seja desligado, ou ocorra algum tipo de erro, ou ainda, que seja encontrada uma instrução de parada. Em outras palavras, a UCP é projetada e fabricada com o propósito único de executar sucessivamente pequenas operações matemáticas (ou outras manipulações simples com dados), na ordem e na seqüência definidas pela organização do programa.

A Fig. 6.2 mostra uma seqüência simplificada do funcionamento de uma UCP e mesmo do conjunto de etapas que constituem efetivamente o ciclo de uma instrução.

Os termos INÍCIO e TÉRMINO, constantes na figura, podem ser entendidos como o início e término do funcionamento da máquina, isto é, quando se liga a chave de alimentação (*power on*) e quando se desliga o computador (*power off*). Durante todo o tempo em que a máquina está ligada, ela executa ininterruptamente ciclos de instrução, como mostrado na figura a seguir:



Mais adiante, iremos observar que um ciclo de instrução é constituído de etapas mais detalhadas do que as que mostramos até o momento. Por exemplo, antes de realizar a operação o processador deve buscar o(s) dado(s) que será(ão) manipulado(s) durante a execução da operação, quando for o caso de uma operação com dados.

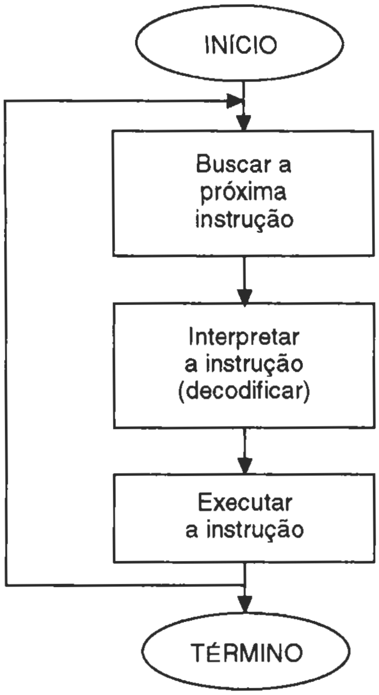

INÍCIO
Buscar a próxima instrução
Interpretar a instrução (decodificar)
Executar a instrução
TÉRMINO

Figura 6.2 Fluxo básico (resumido) de um ciclo de instrução.

As funções realizadas pela UCP

As atividades realizadas pela UCP podem ser divididas em duas grandes categorias funcionais:

- função processamento; e
- função controle.

A Fig. 6.3 mostra o diagrama de blocos de uma UCP simples, com os principais elementos que compõem cada categoria funcional. Naturalmente que se trata de uma organização lógica, funcional e não da organiza-

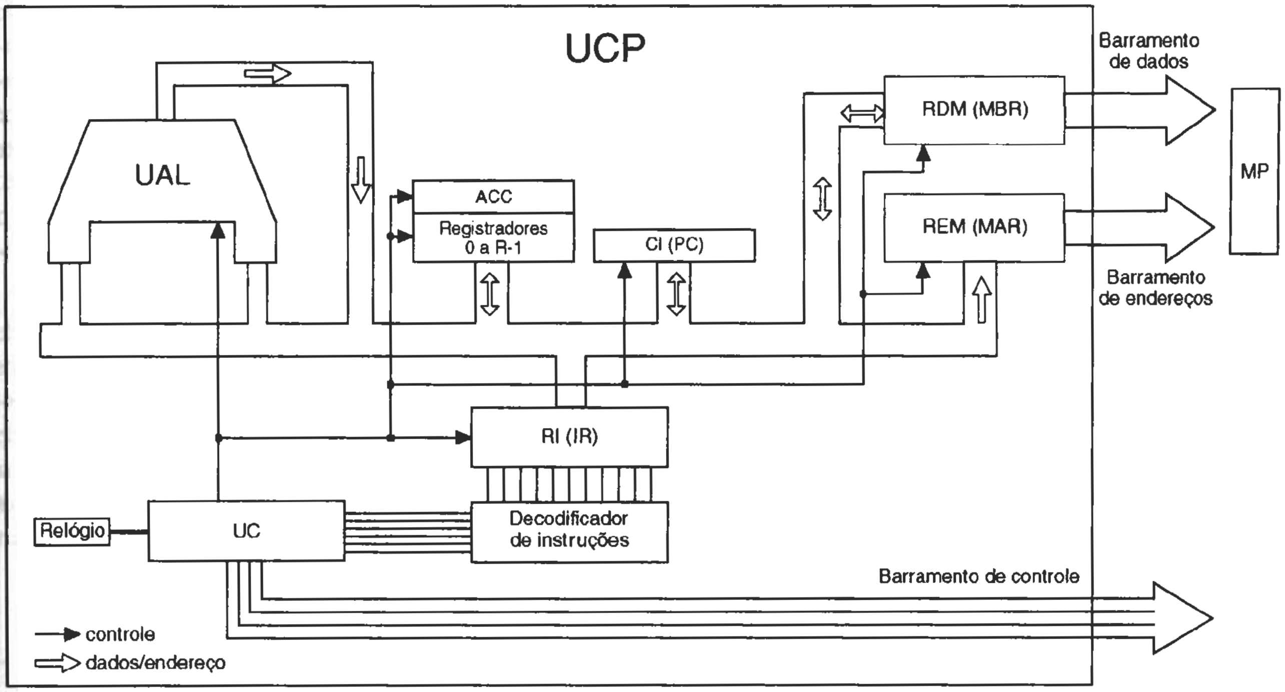

UCP
UAL
ACC
Registradores 0 a R-1
CI (PC)
RDM (MBR)
REM (MAR)
Barramento de dados
Barramento de endereços
MP
RI (IR)
Relógio
UC
Decodificador de instruções
Barramento de controle
controle
dados/endereço

Figura 6.3 Esquema simplificado de uma UCP.

ção física, de como os diversos dispositivos estão fisicamente organizados no interior do processador, o que não interessa ao escopo deste livro.

O diagrama baseia-se em uma UCP de um dos primeiros microprocessadores lançados no mercado, sendo, portanto, bem simples, escolhido justamente para facilitar as primeiras explicações sobre o assunto. Mesmo assim, foram inseridos apenas os elementos básicos, necessários ao entendimento do seu funcionamento.

Ainda dentro do princípio de permitir um melhor entendimento do leitor sobre o processo de realização de um ciclo de instrução nos processadores e, portanto, de como esse componente vital funciona como um todo, escolheu-se uma arquitetura de processador do tipo SISD (ver item 1.1.1), em que os programas são executados de forma seqüencial, isto é, uma instrução é completada, passo a passo, e somente após seu término se inicia a próxima. Tal procedimento torna a execução dos programas lenta, porém nos permite compreender melhor o processo de realização dos ciclos de instrução. As arquiteturas modernas (a partir do processador Intel 8086/8088) executam os ciclos de instrução de forma concorrente (linha de montagem ou *pipelining* — ver item 6.6.4), acelerando o término progressivo das instruções. No entanto, para iniciantes, fica mais difícil a compreensão do funcionamento básico de um ciclo de instrução com essas arquiteturas modernas em vez da antiga e mais simples SISD dos processadores Intel 8080/8085.

Um processador atual, como o microprocessador Intel Pentium ou Motorola/IBM/Apple PowerPC, possui muitos outros elementos complementares, porém os que estão mostrados no diagrama da Fig. 6.3 permanecem válidos e são essenciais para dar maior clareza às explicações que se seguem. No item 6.6.1 são mostrados outros diagramas, mais complexos, de processadores modernos e atuais, com suas características, e ali o leitor poderá conhecer o funcionamento desses processadores.

Descrevendo as duas funções citadas anteriormente, pode-se afirmar que a *função processamento* se encarrega de realizar as atividades relacionadas com a efetiva execução de uma operação, ou seja, processar. O item 6.2.1 apresenta os detalhes mais importantes relativos aos componentes da UCP que constituem esta área.

A *função controle* é exercida pelos componentes da UCP que se encarregam das atividades de busca, interpretação e controle da execução das instruções, bem como do controle da ação dos demais componentes do sistema de computação (memória, entrada/saída). O item 6.2.2 descreve em detalhes os elementos desta área da UCP e suas atividades.

Conceitualmente, podemos imaginar que uma UCP simples, sem elementos mais complexos, pode ser dividida nestas duas áreas. Esta divisão é bem apropriada para caracterizarmos e entendermos melhor o processo de execução seqüencial e serial de um ciclo de uma instrução. *Processo seqüencial* ou *serial* é aquele típico das máquinas com arquitetura SIMD, no qual cada pequena atividade do ciclo de instrução (ver Fig. 6.1) é realizada em seqüência à anterior (algumas poucas atividades podem ser realizadas simultaneamente). Como já mencionamos mais de uma vez, é um processo lento e pouco eficiente, se desejarmos maior velocidade de processamento. Na maior parte deste capítulo, vamos tratar do funcionamento de uma UCP que funciona de modo seqüencial; o leitor poderá, conforme também já mencionamos, obter informações sobre arquiteturas atuais, mais avançadas, consultando o item 6.6. Além disso, no Cap. 11 há informações e detalhes sobre outro tipo de arquitetura de computadores, denominada RISC — *Reduced Instruction Set Computers* (Computadores com Conjunto Reduzido de Instruções).

Os sistemas de computação modernos são montados em torno de processadores que buscam maior velocidade na realização de suas atividades. Um dos processos mais usados para o aumento de velocidade é o que conhecemos na indústria como *linha de montagem ("pipeline")*, no qual a UCP se divide em várias partes funcionais distintas (estágios), cada uma correspondendo a uma determinada atividade. Dessa forma, várias instruções são realizadas de forma simultânea, embora em estágios diferentes, exatamente como ocorre em uma linha de montagem de uma montadora de automóveis. Os processadores Intel 8086 e 8088 introduziram, na linha de microprocessadores, dois estágios de *pipelining*. Já o processador Intel 80486 atingiu 5 estágios de processamento independente; o Intel Pentium Pro e o PowerPC operam com 6 estágios funcionais; o Intel Pentium II (6.ª geração) trabalha com 10 estágios.

6.2.1 Função Processamento

No Cap. 1 foi apresentado o conceito do que significa processamento de dados, a ação de manipular um ou mais valores (dados) em uma certa seqüência de ações, de modo a produzir um resultado útil. O resultado muda conforme o tipo de operação realizada (ou seja, de acordo com a seqüência de ações — de acordo com a instrução específica). Esta é a essência dos sistemas de computação comerciais, que combinam o hardware, fixo e imutável, capaz de realizar diferentes tarefas conforme a ordem e seqüência de instruções que recebe, com o software.

Por exemplo, se uma instrução define que deve ser realizada uma operação de adição sobre os valores A = 5 e B = 3, o sistema, ao interpretar a instrução, gera as ações subseqüentes que redundarão no resultado de valor igual a 5 + 3 = 8. Por outro lado, se o sistema interpretar uma outra instrução que defina a operação de subtração, ele deve gerar outras ações (embora sobre os mesmos dados) de modo que o resultado seja 5 − 3 = 2 (e não mais 8). É claro que parte das ações realizadas para executar as duas instruções é igual.

Processar o dado é executar com ele uma ação que produza algum tipo de resultado. Esta é, pois, a atividade-fim do sistema; ele existe para processar dados. Entre as tarefas comuns a esta função — processamento — podem ser citadas as que realizam:

- operações aritméticas (somar, subtrair, multiplicar, dividir);
- operações lógicas (and, or, xor etc. — ver Cap. 4);
- movimentação de dados (memória – UCP, UCP – memória, registrador – registrador, etc.);
- desvios (alteração de seqüência de execução de instruções);
- operações de entrada ou saída.

O dispositivo principal desta área de atividades das UCP é chamado de UAL — Unidade Aritmética e Lógica. Os demais componentes relacionados com a função processamento são os registradores, que servem para armazenar dados (ou para guardar resultados) que serão usados pela UAL. A interligação entre estes componentes é efetuada pelo barramento interno da UCP (ver item 6.6.3).

A Fig. 6.4 repete o diagrama em bloco da Fig. 6.3, com a diferença que nesta última os elementos relacionados com a função processamento estão realçados.

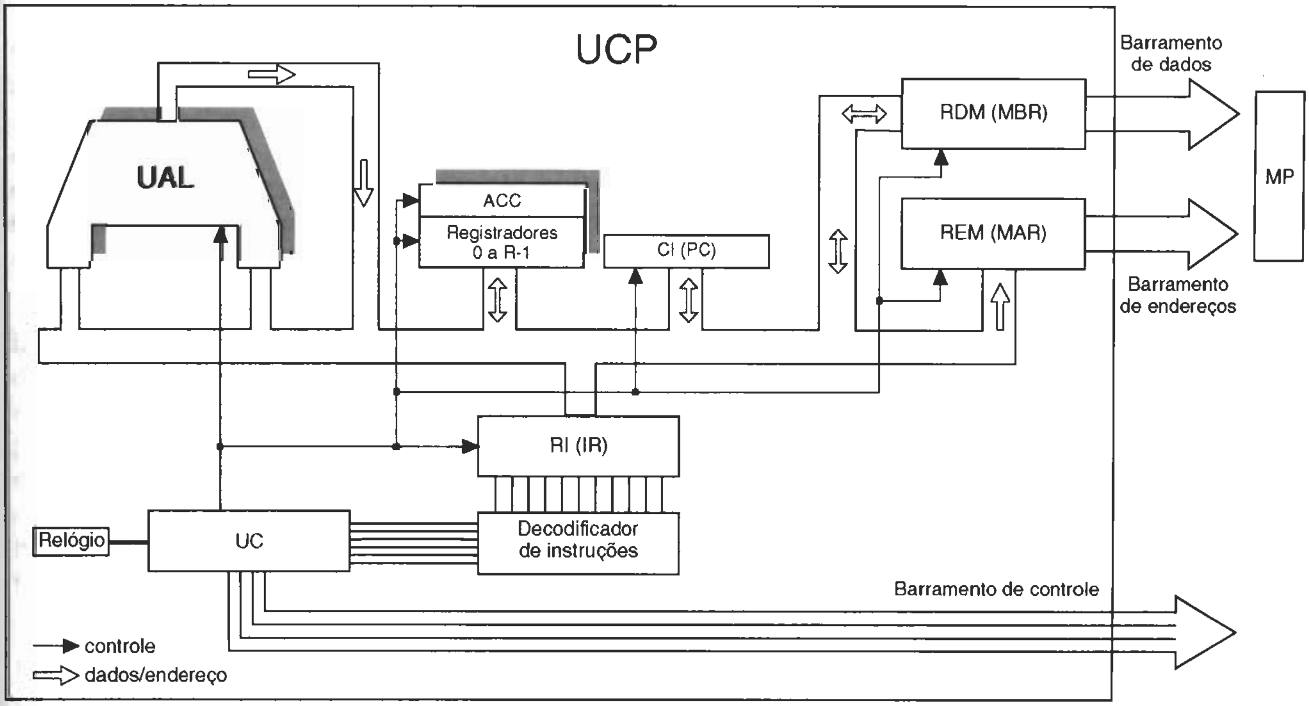


Figura 6.4 Esquema da UCP mostrada na Fig. 6.3, com realce para os elementos que contribuem para a realização da função processamento.

6.2.1.1 Unidade Aritmética e Lógica — UAL

A UAL é o dispositivo da UCP que executa realmente as operações matemáticas com os dados. Tais operações podem ser:

– soma	– subtração
– multiplicação	– divisão
– operação lógica AND	– operação lógica OR
– operação lógica XOR	– operação complemento
– deslocamento à direita	– deslocamento à esquerda
– incremento	– decremento

Tais operações podem utilizar dois valores (operações aritméticas, operações lógicas), por isso a UAL possui duas entradas (ver Figs. 6.3 e 6.4) ou apenas um valor (como, por exemplo, a operação de complemento — ver item 7.5.1). Ambas as entradas se conectam à saída (resultado da operação efetuada) pelo barramento interno de dados. Nos processadores mais antigos, o barramento interno de dados servia para interligar a UAL ao ACC e aos demais registradores e daí à memória principal, conforme pode ser observado nas Figs. 6.3 e 6.4. Nos processadores mais modernos, com mais componentes, o barramento interno conduz os bits de dados de e para a memória cache L1 (e L2 quando ambas são internas, como acontece com o processador Intel Pentium Pro) e daí para as UAL que processam valores inteiros e fracionários (ver Fig. 6.5).

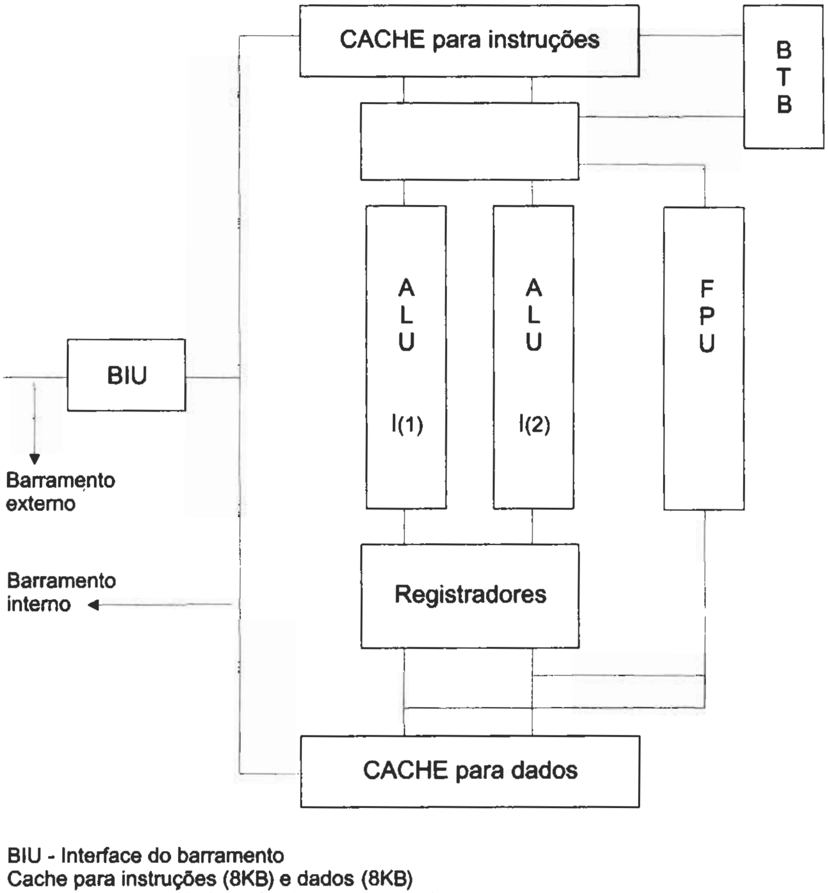

CACHE para instruções
BTB
ALU I(1)
ALU I(2)
FPU
BIU
Barramento externo
Barramento interno
Registradores
CACHE para dados
BIU - Interface do barramento
Cache para instruções (8KB) e dados (8KB)
ALU (1) e (2) - Unidades aritméticas para inteiros

Figura 6.5 Arquitetura simplificada do Pentium original.

A simples observação dos elementos contidos na figura e sua comparação com a arquitetura mostrada nas figuras anteriores indica diversas diferenças, as quais serão descritas com mais detalhes no item 6.6.1. A título de informação inicial, pode-se mencionar entre as diferenças o surgimento de uma memória interna do processador (cache interna — L1), no caso presente, de 16 Kbytes, sendo dividida em dois componentes separados, a cache que somente armazena dados (8KB) e a cache que armazena as instruções (8KB), bem como mais de uma UAL, sendo duas para efetuar cálculos com valores inteiros e uma para efetuar separadamente cálculos com valores fracionários (números representados em ponto flutuante — ver Cap. 7).

Qualquer UAL é um aglomerado de circuitos lógicos (ver Cap. 4) e componentes eletrônicos simples que, integrados, realizam as operações já mencionadas. Ela pode ser uma parte pequena da pastilha do processador, usada em pequenos sistemas, ou pode compreender um considerável conjunto de componentes lógicos de alta velocidade, sendo que os processadores mais modernos utilizam em sua arquitetura mais de uma UAL, de modo a tornar a execução das instruções mais rápida.

A Fig. 6.5, que apresenta uma parte da arquitetura dos processadores Pentium, já mostra essa particularidade, contendo 3 (três) UAL, duas delas para processarem números inteiros e a terceira, denominada FPU — Floating Point Unit, para processar números fracionários. Atualmente, mais UAL são inseridas nos processadores, com o conseqüente aumento de rendimento nas operações matemáticas.

A despeito da grande variação de velocidade, tamanho e complexidade, as operações aritméticas e lógicas realizadas por uma UAL seguem sempre os mesmos princípios fundamentais das UAL mais antigas. No item 6.6.3 são apresentados mais detalhes sobre as UAL e seu funcionamento.

6.2.1.2 Registradores

Para que um dado possa ser transferido para a UAL, é necessário que ele permaneça, mesmo que por um breve instante, armazenado em um registrador (a memória específica da UCP — ver Cap. 5). Além disso, o resultado de uma operação aritmética ou lógica realizada na UAL deve ser armazenado temporariamente, de modo que possa ser reutilizado mais adiante (por outra instrução) ou apenas para ser, em seguida, transferido para a memória. Mesmo no caso dos atuais processadores, dotados de uma quantidade apreciável de espaço de armazenamento nas memórias cache interna, L1, os registradores continuam a existir e ser importantes. Nos processadores com arquitetura do tipo RISC, por exemplo, a importância dos registradores é considerável e sua quantidade, por isso, maior que a dos processadores com arquitetura do tipo CISC, conforme pode ser constatado com detalhe no Cap. 11.

Para atender aos propósitos mencionados, as UCP são fabricadas contendo uma certa quantidade de registradores, destinados ao armazenamento de dados. Servem, pois, de memória auxiliar básica da UAL. Os sistemas mais antigos, como mostrado nas Figs. 6.3 e 6.4, possuíam um registrador especial, denominado *acumulador* (abrevia-se, em inglês, ACC), o qual, além de armazenar dados, servia, também, de elemento de ligação da UAL com os restantes dispositivos da UCP, conforme já explicado. Posteriormente, a partir do surgimento da segunda geração de microprocessadores — os Intel 8086/8088 — o ACC desapareceu, optando-se por inserir no processador registradores especializados no armazenamento de dados, de endereços e de segmentos, cujas funções estão apresentadas no item 6.6.1 (embora seus nomes sejam mais ou menos auto-explicativos).

Em geral, os registradores de dados da UCP têm uma largura (quantidade de bits que podem armazenar) igual ao tamanho estabelecido pelo fabricante para a palavra do referido processador. O tamanho da palavra dos antigos processadores de grande porte IBM/370 era de 32 bits, a mesma largura dos 16 registradores de emprego geral neles existentes. O microprocessador Intel 8088, que moveu os primeiros sistemas IBM PC, possuía registradores de 16 bits cada um, tamanho idêntico ao definido pela Intel para a palavra. Os processadores Pentium de 6.ª geração possuem palavra de 32 bits e 8 registradores de mesmo tamanho, denominados de emprego geral; além disso, possuem também 6 registradores de 16 bits denominados registradores de segmento, que funcionam para o cálculo dos endereços de acesso à memória na execução de programas (ver item 6.6.1).

A quantidade e o emprego dos registradores variam bastante de modelo para modelo de UCP. A Fig. 6.3 mostra o ACC e indica um grupo de registradores de dados, de endereços genéricos R0 a R1; as Figs. 6.6, 6.7 e 6.8 apresentam exemplos de organização de registradores de dados de alguns processadores, Intel 8085 (microprocessador de palavra igual a 8 bits), Intel 8088 (primeiros PC — palavra de 16 bits) e Pentium III (palavra ainda de 32 bits), respectivamente.

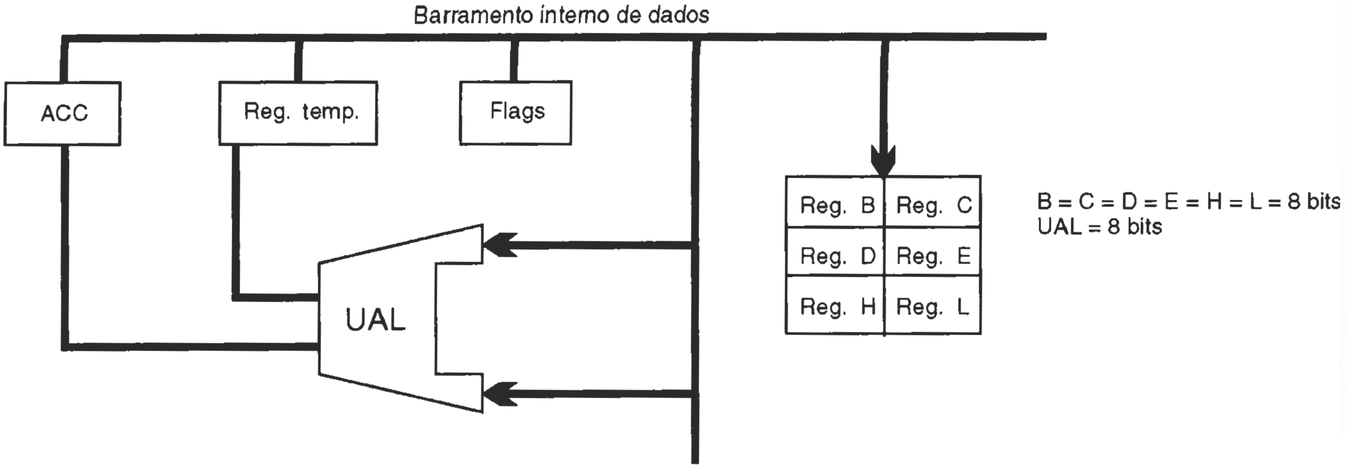

Barramento interno de dados
ACC
Reg. temp.
Flags
UAL
Reg. B
Reg. C
Reg. D
Reg. E
Reg. H
Reg. L
B = C = D = E = H = L = 8 bits
UAL = 8 bits

Figura 6.6 Diagrama (simplificado) em bloco da UCP Intel 8085, apenas com os dispositivos básicos da área de processamento.

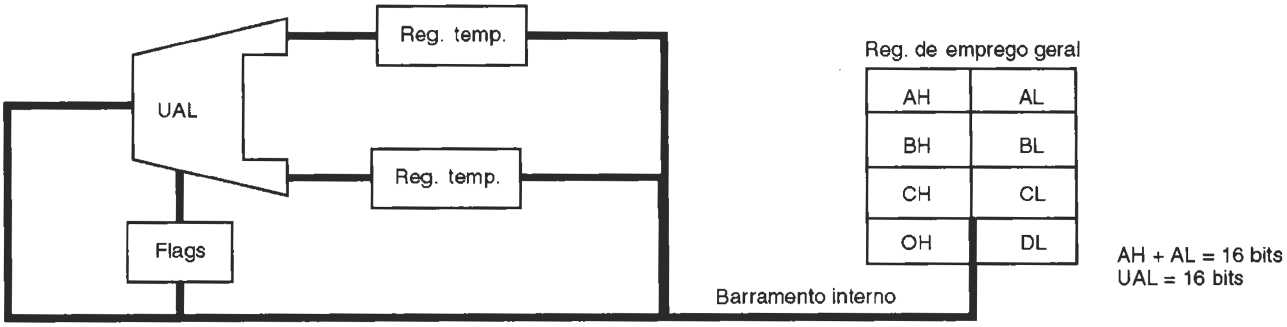

UAL
Reg. temp.
Reg. temp.
Flags
Reg. de emprego geral
AH
AL
BH
BL
CH
CL
OH
DL
Barramento interno
AH + AL = 16 bits
UAL = 16 bits

Figura 6.7 Diagrama (simplificado) em bloco da UCP Intel 8086, apenas mostrando os dispositivos básicos da área de processamento.

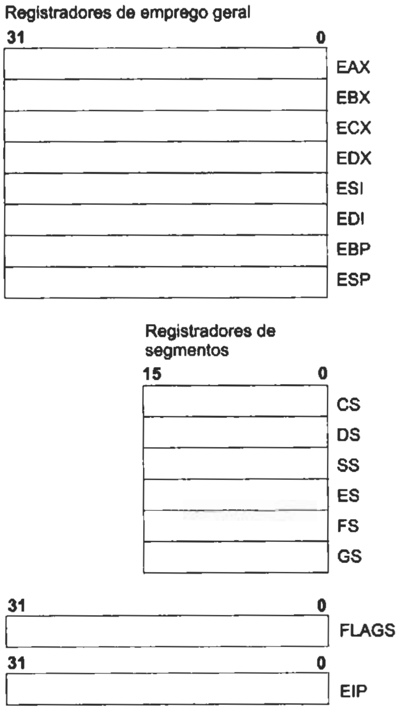

Registradores de emprego geral
31
0
EAX
EBX
ECX
EDX
ESI
EDI
EBP
ESP
Registradores de segmentos
15
0
CS
DS
SS
ES
FS
GS
31
0
FLAGS
31
0
EIP

Figura 6.8 Registradores na arquitetura dos processadores Pentium II.

Além dos registradores de dados, a UCP possui sempre outros registradores (que não participam diretamente da função processamento), com funções específicas ou que funcionam para a área de controle, os quais serão descritos mais adiante. Entre estes registradores podemos citar desde já o registrador de instrução (RI) e o contador de instrução (CI) ou *PC — Program Counter*, que a Intel denomina *EIP — Instruction Pointer*, nos processadores mais modernos (Pentium), além do registrador de endereços de memória (REM) e o registrador de dados de memória (RDM), já descritos no Cap. 5.

Registradores especiais de estado

Além dos registradores usuais para armazenamento de dados e de endereços, a área de processamento se vale de um outro tipo de registradores que auxiliam e completam a realização das operações matemáticas pela UAL, indicando o estado de vários elementos referentes à operação em si. Alguns fabricantes denominam esse conjunto de registradores de *PSW — Program Status Word*, que poderia ser traduzido como palavra de estado do programa.

Na realidade, o registrador se comporta conceitualmente de modo diferente dos demais registradores existentes na UCP, pois, em vez de armazenar dados de forma integral, isto é, um único valor, ele se divide em bits que possuem significado diferente, um por um. Os principais bits de estado, comumente encontrados nos processadores, são:

- sinal — contém o sinal resultante da última operação aritmética realizada pelo processador.
- overflow — quando setado (=1) indica que a última operação aritmética realizada resultou em estouro do valor, um erro.
- zero — quando setado (=1) indica que a última operação aritmética realizada resultou no valor zero.
- vai 1 (*carry*) — indica que ocorreu "vai 1" para o bit mais à esquerda na última operação de soma realizada. Pode indicar, também, overflow em operações com números sem sinal.
- paridade — é setado (=1) ou não (=0), dependendo da quantidade de bits 1 no byte recebido (ver Cap. 7).

A Fig. 6.9(a) mostra o conjunto de bits do registrador de estado dos processadores Pentium, denominados conjuntamente registradores Eflags, enquanto a Fig. 6.9(b) mostra o mesmo tipo de bits de estado, utilizados pelos processadores Motorola, 68000, denominados códigos de condição (*condition codes*).

6.2.1.3 A Influência do Tamanho da Palavra

A capacidade de processamento de uma UCP (a velocidade com que realiza o ciclo de uma instrução) é em grande parte determinada pelas facilidades embutidas no hardware da UAL (ela é, aliás, só hardware) para realizar as operações matemáticas projetadas.[1]

Um dos elementos fundamentais para isso é a definição do *tamanho da palavra* da UCP. O valor escolhido no projeto de fabricação da UCP determinará o tamanho dos elementos ligados à área de processamento, a UAL e os registradores de dados.

Nos processadores mais antigos, como o 8080/8085, 8086/8088 ou mesmo o 80386, também o barramento interno de dados tinha uma largura, em bits, igual ao tamanho da palavra. Com a inserção de uma memória cache interna nos processadores, como no 486, e as L1 dos Intel Pentium e AMD K6, tornou-se mais vantajoso buscar mais dados de cada vez das memórias externas ao processador, já que estes poderiam ser armazenados na cache L1 antes de requeridos pelo processador, acelerando a velocidade de processamento. Desta forma, os barramentos de dados passaram a ter uma largura, em bits, maior do que o tamanho da palavra. Nos Pentium originais, o barramento interno de dados tinha largura de 64 bits, atualmente aumentado para 128 bits.

---

[1] Na realidade, muitos fatores contribuem para o aumento da capacidade de processamento de uma UCP e não somente o tamanho da palavra, como, por exemplo, a quantidade de UAL existentes no processador (atualmente é comum o uso de duas ou mais unidades aritméticas, separando, inclusive, o cálculo com inteiros (ponto fixo) do de fracionários (ponto flutuante) em unidades independentes e especializadas para cada caso).

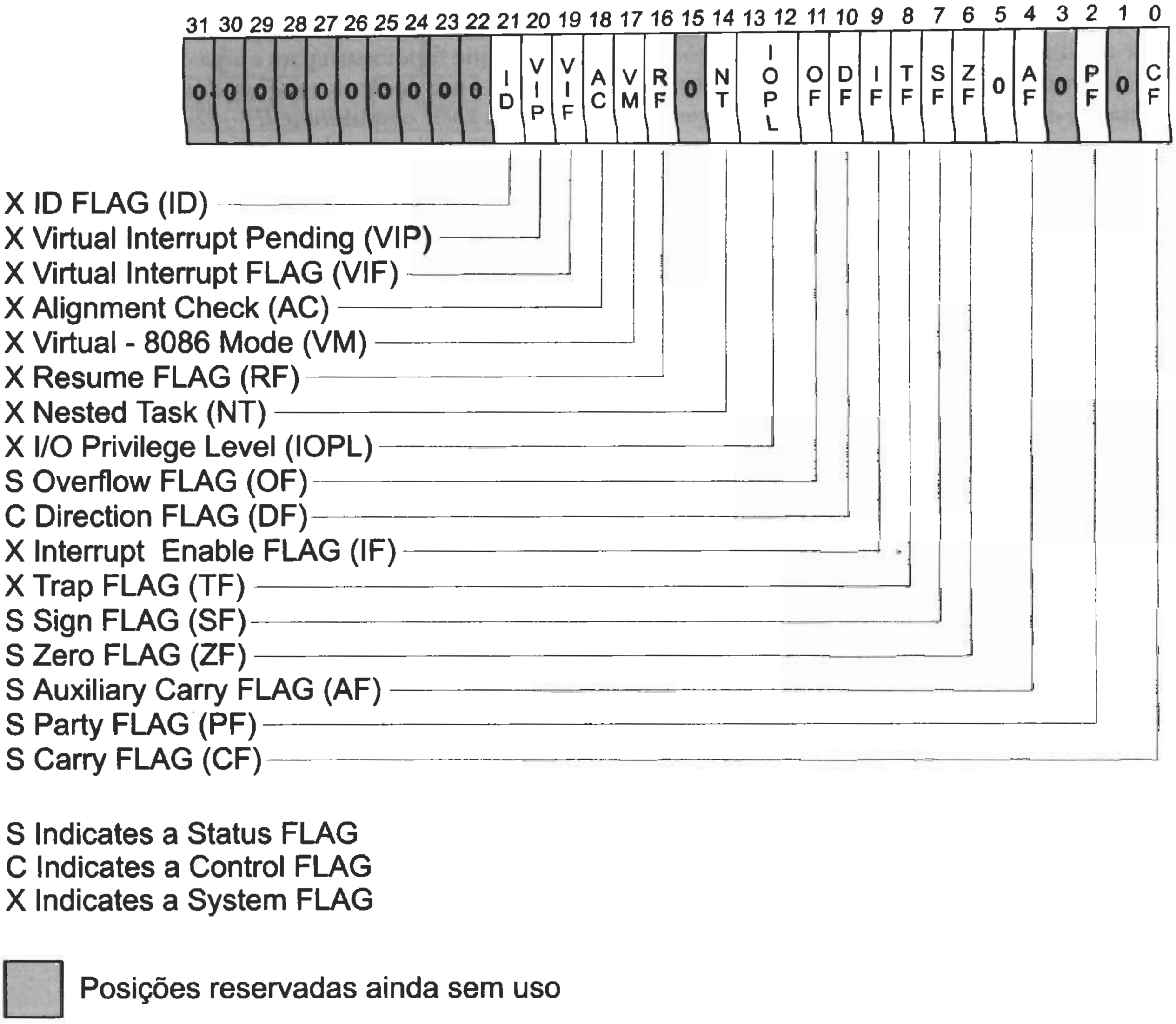
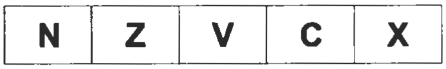

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 ID VIP VIF AC VM RF 0 NT IOPL OF DF IF TF SF ZF 0 AF 0 PF 0 CF
X ID FLAG (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt FLAG (VIF)
X Alignment Check (AC)
X Virtual - 8086 Mode (VM)
X Resume FLAG (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow FLAG (OF)
C Direction FLAG (DF)
X Interrupt Enable FLAG (IF)
X Trap FLAG (TF)
S Sign FLAG (SF)
S Zero FLAG (ZF)
S Auxiliary Carry FLAG (AF)
S Party FLAG (PF)
S Carry FLAG (CF)
S Indicates a Status FLAG
C Indicates a Control FLAG
X Indicates a System FLAG
Posições reservadas ainda sem uso

Figura 6.9(a) Registrador e Flags do processador Pentium.



N - Negative FLAG - Valor 1 se o resultado de uma operação aritmética for negativo.
Z - Zero FLAG - Valor 1 se o resultado for igual a zero.
V - Overflow FLAG - Valor 1 se ocorrer overflow. Isto é detetado se houver troca de serial.
C - Carry FLAG - Valor 1 se ocorrer "vai 1" para fora do número.
X - Extend FLAG - Semelhante ao FLAG C, porém pouco usado.

Figura 6.9(b) Códigos de condição (conditional codes) no processador Motorola 68000.

Um tamanho maior ou menor de palavra (e, por conseguinte, da UAL e dos registradores de dados) acarreta, sem dúvida, diferenças acentuadas de desempenho da UCP.

Com o único propósito de mostrar de forma prática a influência do tamanho da palavra na capacidade de processamento de um processador, vejamos um exemplo simples de cálculo.

Seja, por exemplo, a execução de uma operação de soma de dois valores, A = 3A25 e B = 172C, ambos números inteiros, sem sinal, com 16 bits de tamanho cada um (os dois números estão representados em hexadecimal, sendo por isso armazenados da forma visual mais inteligível, denominada *bi-endian*, isto é, a parte mais significativa no endereço mais baixo e a parte menos significativa no endereço mais alto. Esta forma é utilizada por alguns processadores, mas nem todos. Nos processadores Intel utiliza-se a forma oposta, denominada *little-endian* (no item 6.6.3 descreve-se essas duas formas de armazenamento de dados com um pouco mais de detalhe). A referida soma será simulada em dois sistemas de computação, sistema 1 e sistema 2.

O sistema 1 possui palavra de 8 bits, e a memória principal tem 64K células de 8 bits cada uma, conforme mostrado na Fig. 6.6 (poderia ser, por exemplo, semelhante à arquitetura básica dos processadores Intel 8080/8085 ou Motorola 6800).

O sistema 2 possui palavra de 16 bits, e a memória principal um espaço de endereçamento de 1M células, todas também com 8 bits cada uma, conforme mostrado na Fig. 6.7 (poderia ser, por exemplo, semelhante à arquitetura básica do processador Intel 8086 ou 8088).

Em resumo, a unidade de processamento (este termo "unidade" não é utilizado na literatura nem pelos fabricantes, mas acredito ser pertinente para o entendimento do assunto), a palavra, é diferente nos dois sistemas, sendo a do sistema 2 o dobro do tamanho da do sistema 1, enquanto a unidade de armazenamento de ambos os sistemas tem o mesmo valor, 8 bits. A título de informação pode-se afirmar que ainda nos processadores mais modernos a unidade de armazenamento continua sendo o byte, isto é, as células são organizadas de modo a cada uma armazenar um byte de dados; enquanto isso, o tamanho da palavra vem evoluindo sistematicamente do valor inicial de 4 bits do primeiro microprocessador (Intel 4004) para 64 bits dos processadores mais modernos e para o futuro (o Alpha, da Compaq/DEC, o Merced, da Intel e o Athlon da AMD, por exemplo).

Operação de soma no sistema 1

a) A operação é realizada em duas etapas lógicas (na prática, são gastos diversos tempos de relógio, conforme será mostrado no item 6.4), porque cada valor tem 16 bits e a UCP (UAL, registrador ACC e barramento de dados) só permite armazenar, processar e transferir dados com 8 bits de tamanho.

b) Na primeira etapa é transferida para a UAL, via ACC e barramento de dados, a 1.ª metade de cada número (25 para o número A e 2C para o número B) e eles são somados.

c) Na segunda etapa a operação é realizada de forma idêntica, exceto para a 2.ª parte dos valores (3A para o número A e 17 para o número B).

d) A operação completa gasta um período de tempo igual a T_1 (soma dos tempos $T_1/2$ da etapa 1 e $T_1/2$ da etapa 2).

A Fig. 6.10 mostra este exemplo em um diagrama em bloco de uma UCP semelhante à do sistema 1, com a transferência dos valores sendo efetuada de 8 em 8 bits de cada vez.

Operação de soma no sistema 2

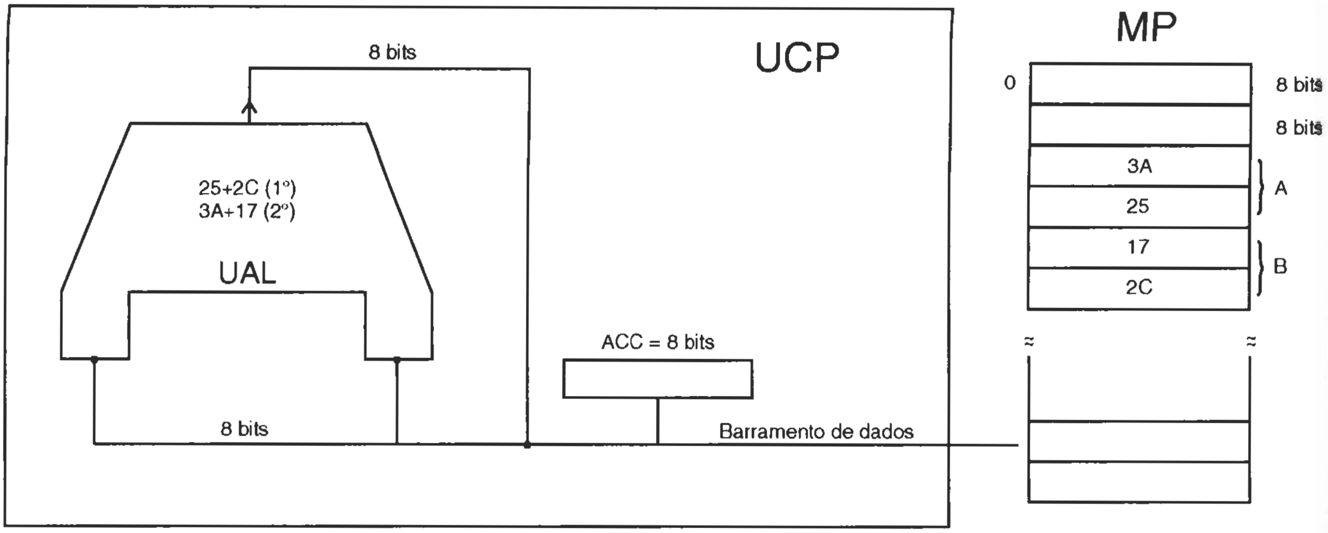
a) A operação é realizada em uma única etapa lógica, porque a UCP é fabricada para operar valores de 16 bits de tamanho, mesmo tamanho dos números. Desse modo, os números não necessitam ser divididos em duas partes como no sistema 1.

b) A operação completa gasta um período de tempo igual a T_2.

A Fig. 6.11 mostra este exemplo em um diagrama em bloco de uma UCP semelhante à do sistema 2, com a transferência dos valores sendo efetuada de 16 em 16 bits de cada vez.

Considerando que a operação de soma no sistema 1 é realizada em duas etapas e a mesma operação no sistema 2 é realizada em uma etapa, o tempo T_2 deve ser aproximadamente a metade do tempo T_1. Isto torna a capacidade da UCP do sistema 2 bem maior que a capacidade do sistema 1.

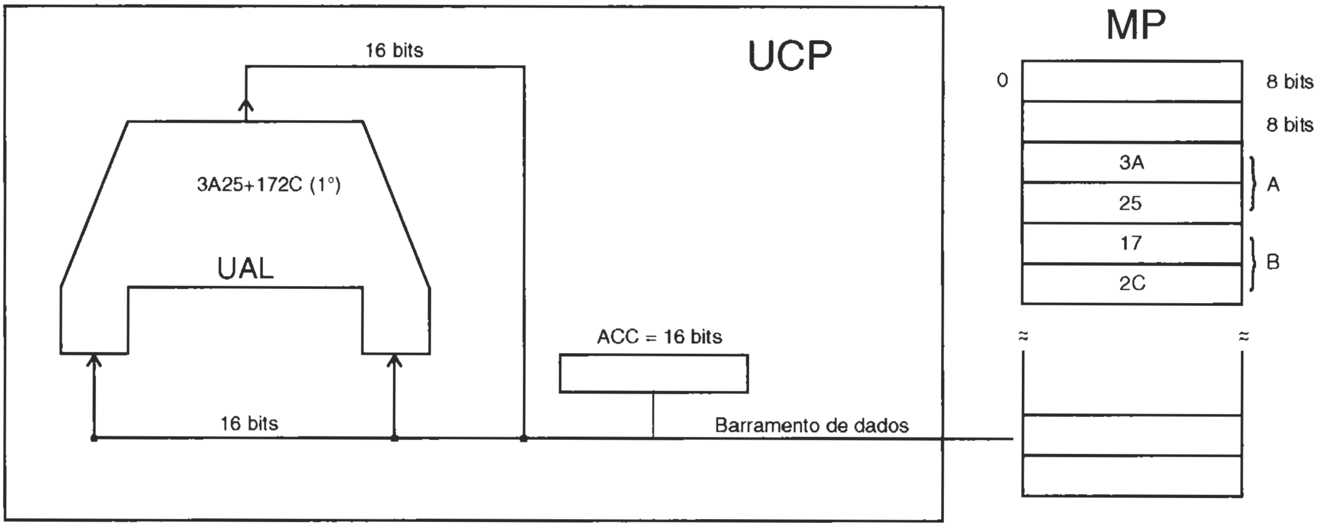
NOTA: É importante ressaltar que, nos exemplos anteriores, foram consideradas várias simplificações, não somente na arquitetura dos dois sistemas, como também no processo de soma, visando não complicar a explicação do essencial — a influência do tamanho da palavra na variação da capacidade de processamento dos sistemas. Nenhuma das simplificações feitas comprometeu a conclusão final.


8 bits
UCP
MP
0
8 bits
8 bits
3A
25
17
2C
A
B
25+2C (1º)
3A+17 (2º)
UAL
ACC = 8 bits
8 bits
Barramento de dados

UCP - palavra de 8 bits
Memória de 64 Kbytes

Operação: somar 2 números com 16 bits de tamanho
A = 3A25 e B = 172C (realizada em duas etapas)

Figura 6.10 Exemplo de uma operação de soma de 2 números, A e B, em um computador com palavra de 8 bits.


16 bits
UCP
MP
0
8 bits
8 bits
3A
25
17
2C
A
B
3A25+172C (1º)
UAL
ACC = 16 bits
16 bits
Barramento de dados

UCP - palavra de 16 bits
Memória de 1 Mbytes

Operação: somar 2 números com 16 bits de tamanho
A = 3A25 e B = 172C (realizada em uma única etapa)

Figura 6.11 Exemplo de uma operação de soma de 2 números, A e B, em um computador com palavra de 16 bits.

No projeto de uma UCP, a definição do tamanho da palavra tem enorme influência no desempenho global de toda a UCP e, por conseguinte, do sistema como um todo. Vejamos:

a) Influência ou desempenho devido ao maior ou menor tempo na execução de instruções com operações matemáticas na UAL, conforme demonstrado no exemplo anterior.

b) Influência no desempenho devido ao tamanho escolhido para o barramento interno e externo da UCP. Conforme já mencionamos, obtém-se o máximo de desempenho quando a largura (tamanho) do barramento de dados é, no mínimo, igual ao tamanho da palavra (como nos exemplos mostrados). Se a largura do barramento for, por exemplo, igual a 16 bits em um sistema com palavra de 32 bits (UAL e registradores de 32 bits), então o movimento de 4 bytes de um dado tipo de caractere requererá dois ciclos de tempo do barramento, ao passo que em barramento de 32 bits requereria apenas um ciclo de tempo.

c) Influência também na implementação física do acesso à memória. Embora atualmente a capacidade das memórias seja medida em bytes (porque as células são sempre de largura igual a 8 bits), o movimento de dados entre UCP e memória é normalmente medido em palavras, porque o barramento de dados que une o RDM (MBR) à memória deve acompanhar em largura o valor da palavra. Para uma UCP de 32 bits de palavra, por exemplo, é desejável que a memória seja organizada de modo que sejam acessadas 4 células contíguas (4 bytes = 32 bits) em um único ciclo de memória. Se isto não ocorrer, a UCP deverá ficar em estado de espera (*wait state*). Como observamos, atualmente, o barramento de dados é ainda maior que o tamanho da palavra, em geral, múltiplo já do tamanho da palavra, acarretando um desempenho ainda melhor do sistema.

Para maiores detalhes sobre fatores que afetam o desempenho de um sistema UCP/ME consultar o item 6.6 — "Um pouco mais de detalhe". A Tabela 6.1 apresenta uma relação de processadores com vários dados característicos, entre os quais o tamanho da palavra.

6.2.2 Função Controle

Já verificamos que as instruções de máquina que compõem um programa em execução devem estar armazenadas seqüencialmente na memória principal (e na cache, se houver uma). Já verificamos também que a UAL é o dispositivo da UCP responsável por realizar a operação matemática definida pela instrução que estiver sendo executada no momento. Falta conhecermos mais detalhes sobre as instruções de máquina, a saber:

a) o que é e como funciona uma instrução de máquina (ver os itens 6.3 e 6.4);

b) como a referida instrução de máquina é movimentada da memória para a UCP;

c) onde a instrução de máquina será armazenada na UCP; e

d) como será identificada e controlada a ação (a operação) que deve ser realizada.

A resposta para estas indagações está no conhecimento das funções desempenhadas pela área de controle de uma UCP.

A área de controle de uma UCP é a parte funcional que realiza (uma etapa de cada vez em sistemas de execução seqüencial, ou várias etapas simultaneamente, em sistemas de execução *pipelining*) as atividades de:

a) busca da instrução que será executada, armazenando-a em um registrador especialmente projetado para esta finalidade;

b) interpretação das ações a serem desencadeadas com a execução da instrução (se é uma soma, uma subtração, uma complementação etc. e como realizá-las). Em inglês, estas duas etapas compõem o que se denomina *fetch cycle*, ou ciclo de busca da instrução; e

c) geração dos sinais de controle apropriados para ativação das atividades requeridas para a execução propriamente dita da instrução identificada. Esses sinais de controle são enviados aos diversos componentes do sistema, sejam internos da UCP (como a UAL) ou externos (como a memória ou E/S). Esta etapa é denominada em inglês *execute cycle* ou ciclo de execução da instrução.

Em outras palavras, a área de controle é projetada para entender o que fazer, como fazer e comandar quem vai fazer no momento adequado. Podemos fazer uma analogia com os seres humanos, imaginando que a área de controle é o cérebro que comanda o ato de andar, e a área de processamento são os músculos e ossos das pessoas que realizam efetivamente o ato. Os nervos são análogos ao barramento de interligação entre os diversos elementos.

A Fig. 6.12 apresenta o mesmo diagrama em bloco da UCP exemplificada na Fig. 6.3, com realce para os dispositivos que fazem parte da área de controle da referida UCP.

Os dispositivos básicos que devem fazer parte daquela área funcional são:

- unidade de controle (UC);
- decodificador;
- registrador de instrução (RI) ou IR — *instruction register*;
- contador de instrução (CI) ou PC — *program counter*;
- relógio ou *clock*;
- registradores de endereço de memória (REM) e de dados da memória (RDM).

A quantidade, a complexidade e a disposição dos componentes que realizam as funções de controle variam consideravelmente de UCP para UCP, porém, essencialmente, os dispositivos indicados (realçados na Fig. 6.12) são os mesmos.

Com a finalidade de explicar de modo mais simples o funcionamento da UCP, continuaremos a adotar o esquema das Figs. 6.3, 6.4 e 6.12.

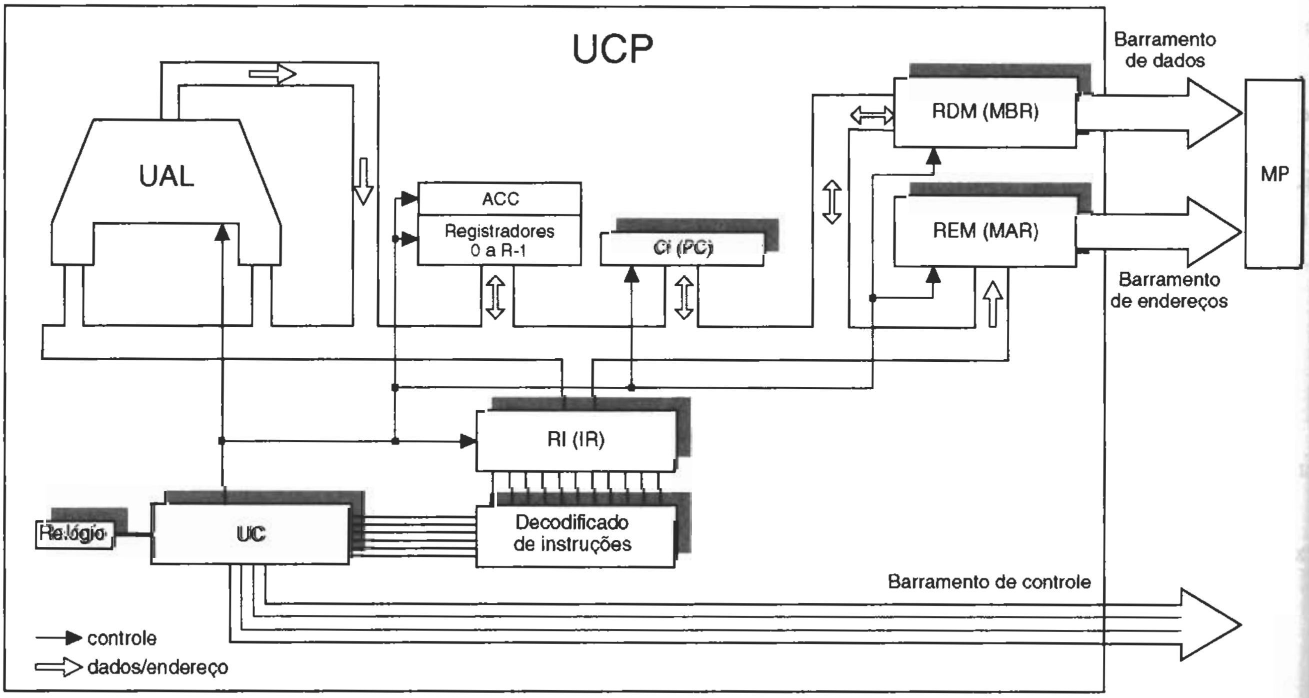

UCP
UAL
ACC
Registradores 0 a R-1
CI (PC)
RDM (MBR)
REM (MAR)
Barramento de dados
MP
Barramento de endereços
RI (IR)
Relógio
UC
Decodificado de instruções
Barramento de controle
controle
dados/endereço

Figura 6.12 Esquema da UCP mostrada na Fig. 6.3, com realce para os elementos que contribuem para a realização da função controle.

6.2.2.1 A Unidade de Controle

É o dispositivo mais complexo da UCP. Ele possui a lógica necessária para realizar a movimentação de dados e de instruções de e para a UCP, através dos sinais de controle que emite em instantes de tempo programados. A Fig. 6.13 mostra um diagrama em bloco simplificado dos principais elementos da função controle de um processador, ressaltando a Unidade de Controle, UC. Como também se observa na Fig. 6.12, a UC se conecta a todos os principais elementos do processador e ao barramento de controle, como, por exemplo, a UAL. Os sinais de controle, emitidos pela UC (ver Fig. 6.3), ocorrem em vários instantes durante o período de realização de um ciclo de instrução e, de modo geral, todos possuem uma duração fixa e igual, originada em um gerador de sinais denominado relógio (*clock*).

Os microeventos ou microoperações comandadas pelo funcionamento da UC podem ser iniciados segundo um de dois princípios de arquitetura (ver item 6.6.4):

- por microprogramação; ou
- por programação prévia diretamente no hardware.

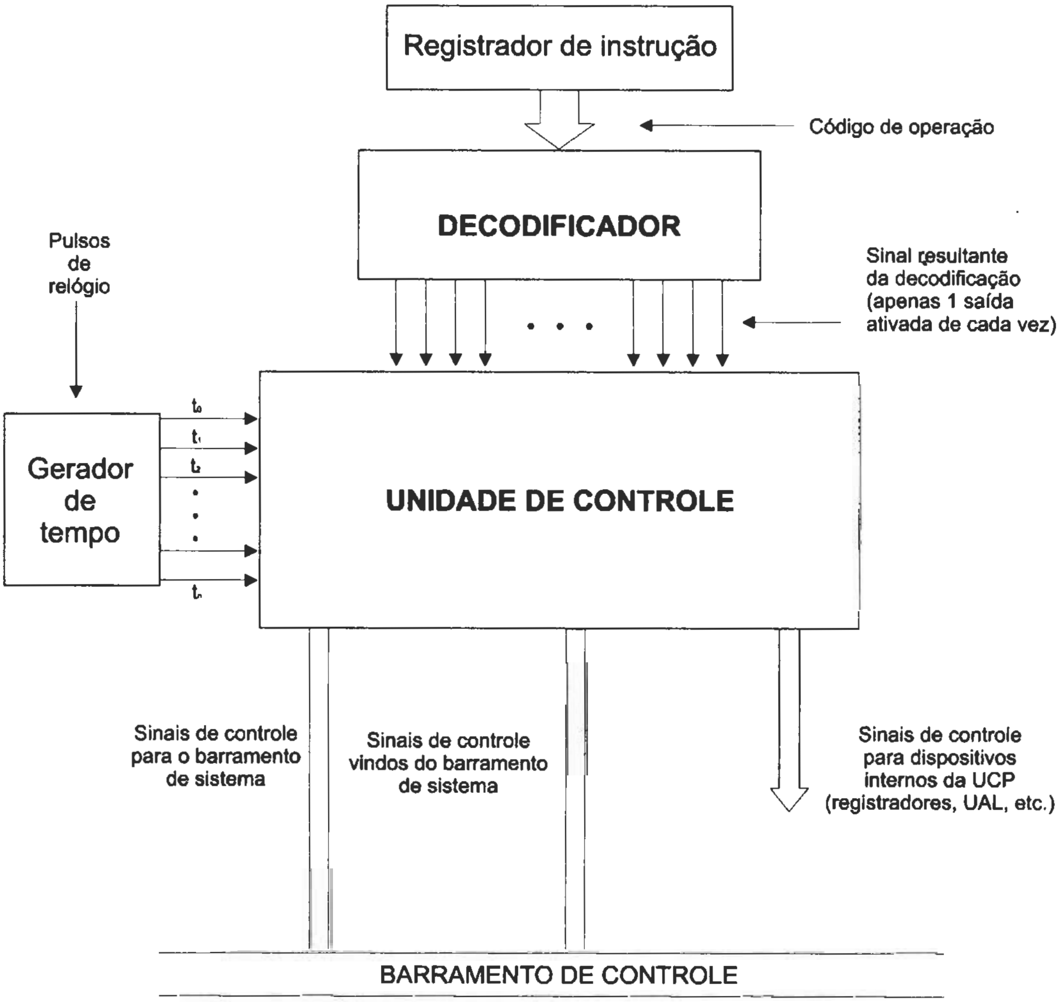

Registrador de instrução
Código de operação
DECODIFICADOR
Pulsos de relógio
Sinal resultante da decodificação (apenas 1 saída ativada de cada vez)
Gerador de tempo
t0
t1
t2
tn
UNIDADE DE CONTROLE
Sinais de controle para o barramento de sistema
Sinais de controle vindos do barramento de sistema
Sinais de controle para dispositivos internos da UCP (registradores, UAL, etc.)
BARRAMENTO DE CONTROLE

Figura 6.13 Diagrama em bloco simplificado da função controle.

Assim, por exemplo, o início de um ciclo de instrução consiste em buscar (*fetch*) a referida instrução e trazer uma cópia sua da MP para o processador (para o registrador de instrução, como será explicado mais adiante). Para efetivar esta ação são realizadas algumas ações menores que, em conjunto, constituem a desejada transferência (na realidade, constituem os passos de um ciclo de leitura, conforme descrito no item 5.3.3.1). Tais operações menores denominam-se microoperações, por se constituírem na menor parte individualmente executável pelo processador. A Fig. 6.14 mostra as microoperações realizadas para completar o referido ciclo de busca.

Cada microoperação é realizada por iniciativa de um pulso originado na UC em decorrência de uma prévia programação (diretamente no hardware ou pela execução de uma microinstrução, se a arquitetura do processador é microprogramada).

t_0: **REM** ← **(CI)**
t_1: **CI** ← **CI + N**
RDM ← **M(op.)**
t_2: **RI** ← **RDM**

Sendo: t_0, t_1 e t_2 - pulsos de relógio emitidos em instantes seqüencialmente crescentes
REM - registrador de endereços de memória
RDM - registrador de dados de memória
CI - contador de instruções
M(op.) - conteúdos de célula(s) de endereço igual a op.

Figura 6.14 Seqüência de microoperações realizadas em um ciclo de busca (fetch).

Uma outra característica de funcionamento dos sistemas de computação na área de controle, mais especificamente com relação à Unidade de Controle, refere-se ao modo pelo qual o sistema conduz a execução das instruções, redundando em diversos tipos de arquitetura de processadores, tais como:

- processadores que executam instruções de modo exclusivamente seqüencial ou serial (SISD);
- processadores que executam instruções de modo concorrente, ou tipo pipeline ou por linha de montagem;
- processadores que executam várias instruções simultaneamente ou por processamento paralelo;
- processadores que realizam processamento vetorial.

Os dois primeiros tipos serão abordados ainda neste capítulo, sendo os demais apenas mencionados mas não descritos neste texto.

6.2.2.2 O Relógio

É o dispositivo gerador de pulsos cuja duração é chamada de ciclo. A quantidade de vezes em que este pulso básico se repete em um segundo define a unidade de medida do relógio, denominada *freqüência*, a qual também usamos para definir velocidade na UCP. A Fig. 6.15 mostra um exemplo de um relógio (em geral, é um gerador de cristal de quartzo) e os pulsos por ele gerados. Um ciclo de relógio ou de máquina (*machine cycle*) é o intervalo de tempo entre o início de um pulso e o início do seguinte. Este ciclo está relacionado à realização de uma operação elementar, durante o ciclo de uma instrução. No entanto, mesmo esta operação elementar não se realiza em um só passo e, por essa razão, costuma-se dividir o ciclo de máquina em ciclos menores (subciclos), defasados no tempo, de modo que cada um aciona um passo diferente da operação elementar. Esses diferentes passos de uma operação elementar denominam-se microoperações, conforme já mencionamos anteriormente. A Fig. 6.15 mostra o ciclo básico e os cinco subciclos gerados por um retardador (exemplo dos ciclos do processador Intel 8085).

Na Fig. 6.14 apresentamos um exemplo de microoperações realizadas para completar a busca de uma instrução da MP para o Registrador de Instrução, RI, na UCP. Cada microoperação é realizada em um instante de tempo T_n, conforme se observa na figura. Esses instantes de tempo são originados no relógio.

Se as operações, para realizar um ciclo de instrução, duram o tempo definido por um ou mais pulsos do relógio, e se estes pulsos tiverem curta duração, mais operações podem ser realizadas na mesma unidade de tempo (o período-base utilizado é o segundo, ciclos por segundo, milhões de instruções por segundo, etc.).

A unidade de medida utilizada para a freqüência dos relógios de UCP é o hertz (Hz), que significa 1 ciclo por segundo — mesma unidade de medida de freqüência de sinais analógicos, como a voz. Como se trata de freqüências elevadas, abreviam-se os valores usando-se milhões de hertz, ou de ciclos por segundo (megahertz, ou simplesmente MHz).

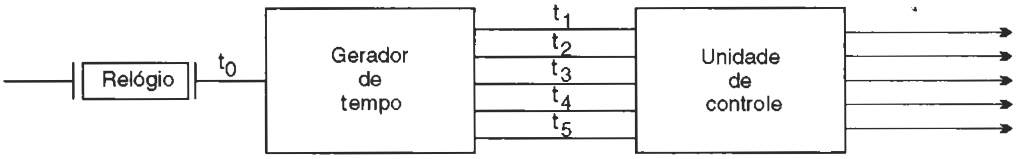
Assim, por exemplo, se um determinado processador funciona com o seu relógio oscilando 25 milhões de vezes por segundo, sua freqüência de operação é de 25 MHz. E como a duração de um ciclo, seu período, é o inverso da freqüência, então cada ciclo, neste exemplo, será igual ao inverso de 25.000.000, ou 1/25.000.000, que é, medido em tempo, 0,00000004 s ou 40 nanossegundos.

Uma UCP com freqüência de 400 MHz possui um período de relógio de 1/400.000.000 s ou 2,5 ns. Este período é o tempo T, mostrado na Fig. 6.15.

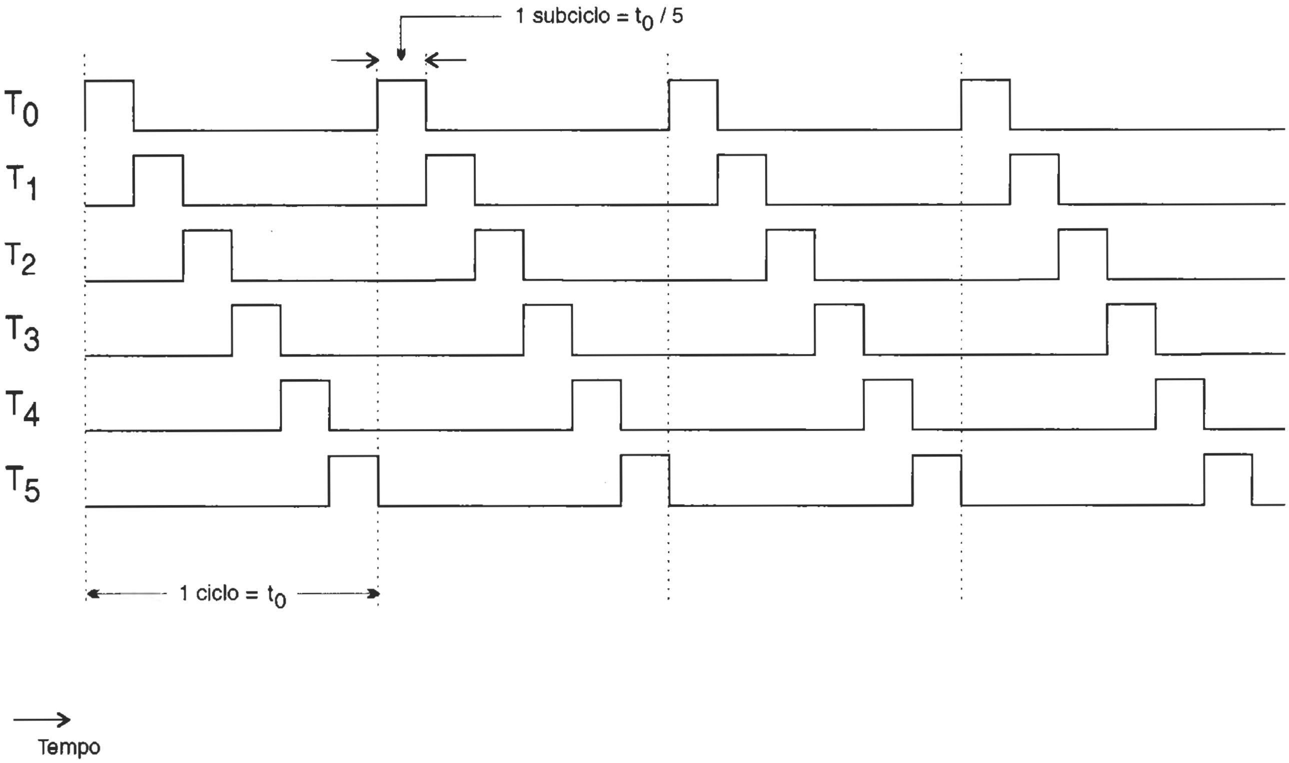
Uma das características de processadores mais conhecida dos usuários que trabalham com computadores ou pretendem adquirir um é justamente a freqüência do relógio da UCP. Ela realmente pode ser considerada também um indicador de desempenho, menos técnico para os leigos. No entanto, não é absolutamente verdade que um processador com velocidade de relógio maior que o outro seja mais eficiente que aquele. Isto porque, se é verdade que a maior velocidade de relógio implica pulsos de duração menores, a tecnologia e a arquitetura de projeto do processador podem torná-lo mais eficiente que um outro, mesmo que funcionando com velocidade de relógio menor.

O processador Pentium possui uma arquitetura superior ao do processador 80486, com mais estágios de *pipelining*, memória cache maior, entre outras características melhores que as do 486. Neste caso, um Pentium funcionando a 66 MHz terá desempenho superior a um 486 que pudesse funcionar a uma velocidade de 80 ou 100 MHz (trata-se apenas de suposição para tornar o exemplo mais claro).

O processador Intel 8088 foi lançado com velocidade de 4,77MHz, enquanto atualmente temos processadores funcionando em velocidades muitas vezes maiores, como o Pentium II, o AMD K6 e o Cyrix M1. A Tabela 6.1 apresenta uma relação mais completa de processadores e suas freqüências de relógio.



(a) Diagrama em bloco do conjunto de tempo da área de controle



(b) Diagrama de tempo do ciclo do processador (t_0) e seus 5 subciclos

Figura 6.15 Diagrama em bloco da UC, mostrando o relógio e um conjunto de ciclos de tempo.

6.2.2.3 Registrador de Instrução (RI) — Instruction Register (IR)

É o registrador (mostrado na Fig. 6.12) que tem a função específica de armazenar a instrução a ser executada pela UCP. Ao se iniciar um ciclo de instrução (ver item 6.4), a UC emite sinais de controle em seqüência no tempo, de modo que se processe a realização de um ciclo de leitura para buscar a instrução na memória (uma cópia dela). Conforme definido na programação do ciclo de busca de um ciclo de instrução, ao término deste ciclo de leitura a instrução desejada será armazenada no RI, via barramento de dados e RDM (no item 6.4 são apresentados mais detalhes da execução completa de um ciclo de instrução).

6.2.2.4 Contador de Instrução (CI) — Program Counter (PC)

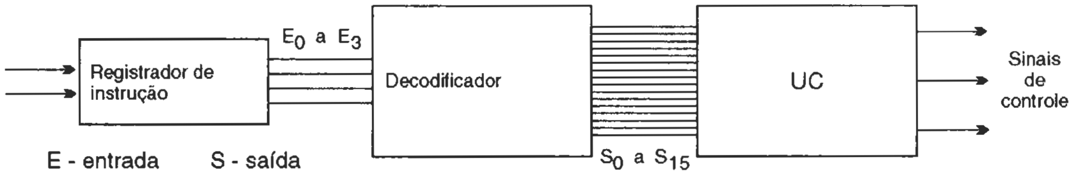
É o registrador cuja função específica é armazenar o endereço da próxima instrução a ser executada. Tão logo a instrução que vai ser executada seja buscada (lida) da memória para a UCP (início do ciclo de instrução), o sistema automaticamente efetiva a modificação do conteúdo do CI de modo que ele passe a armazenar o endereço da próxima instrução na seqüência (ver item 6.4). Por isso, é comum definir a função do CI como a de "armazenar o endereço da próxima instrução"; na realidade, durante a maior parte da realização de um ciclo de instrução, o CI contém o endereço já da próxima instrução.

O CI é um registrador crucial para o processo de controle e de seqüenciamento da execução dos programas. Esta característica será detalhadamente analisada no item 6.4, quando será apresentada, passo a passo, a execução de um ciclo de instrução.

6.2.2.5 Decodificador de Instrução

É um dispositivo utilizado para identificar que operação será realizada, correlacionada à instrução cujo código de operação foi decodificado. Em outras palavras, cada instrução é uma ordem para que a UCP realize uma determinada operação (ver item 6.3). Como são muitas instruções, é necessário que cada uma possua uma identificação própria e única. A unidade de controle está, por sua vez, preparada para sinalizar adequadamente aos diversos dispositivos da UCP, conforme ela tenha identificado a instrução a ser executada.

O decodificador recebe em sua entrada um conjunto de bits previamente escolhido e específico para identificar uma instrução de máquina (cada instrução tem um valor próprio, denominado código de operação, conforme será mostrado no item 6.3), e possui 2^N saídas, sendo N a quantidade de algarismos binários do valor de entrada.



(a) Diagrama em bloco da decodificação em uma UCP

E_0	E_1	E_2	E_3	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(b) Linhas de Saída

Figura 6.16 Exemplo de um decodificador com 4 entradas e 16 saídas.

A Fig. 6.16 mostra um exemplo de configuração de decodificador com entrada de 4 bits e 16 saídas. Cada linha de saída aciona de modo diferente a UC e esta, por sua vez, emite sinais de controle por diferentes caminhos, conforme a linha de saída decodificada. A Fig. 4.5 mostra um exemplo do emprego de um sinal de controle para efetivar a abertura de portas lógicas, transferindo dados de um ponto para outro do sistema.

6.2.2.6 Registrador de Dados de Memória — RDM e Registrador de Endereços de Memória — REM

São os registradores utilizados pela UCP e memória para comunicação e transferência de informações, conforme já explicado no Cap. 5. Eles serão novamente mencionados durante a descrição de um ciclo de instrução, no item 6.4. Em geral, o RDM (ou *MBR — Memory Buffer Register*) possui um tamanho (capacidade de armazenamento de bits) igual ao do barramento de dados; este, atualmente, tem sido construído com largura, em bits, múltipla do tamanho da palavra do processador. Assim é que o Pentium II, por exemplo, possui palavra de 32 bits e seu barramento de dados tem uma largura de 64 bits.

O REM (ou *MAR — Memory Address Register*) possui um tamanho igual ao dos endereços da memória (e, conseqüentemente, do barramento de endereços do sistema). Pela definição do tamanho em bits do REM podemos calcular qual o espaço máximo de endereçamento da memória principal de um computador.

6.3 INSTRUÇÕES DE MÁQUINA

6.3.1 O que É uma Instrução de Máquina?

Desde os primeiros capítulos deste livro, as instruções de máquina vêm sendo mencionadas, referenciadas e utilizadas em uma explicação ou descrição qualquer, sem que tenhamos, até este momento, nos detido a entender efetivamente o que é uma instrução de máquina.

A arquitetura básica dos processadores do tipo von Neumann (ver item 2.1), cujos princípios fundamentais ainda são válidos (a seqüencialidade da execução das instruções deixou de existir com o advento das técnicas *pipelining*, ou de processamento paralelo, ou vetorial, etc.), é essencialmente calcada na existência de uma ordem ou instrução para que o processador (o hardware) realize uma determinada operação (assim como nós instruímos um funcionário a realizar uma determinada atividade — a instrução, aqui, não no sentido de ensinar, mas de comandar a realização de um ato).

Uma máquina pode executar tarefas complicadas e sucessivas se for "instruída" sobre o que fazer e em que seqüência isso deve ser feito. Os seres humanos (pelo menos a maioria), ao receberem uma instrução do tipo "trazer a pasta da funcionária Maria", são capazes de localizar o arquivo em que as pastas de todos os funcionários estão arquivadas — em geral por ordem alfabética — e achar a pasta, trazendo-a a quem pediu. Nosso cérebro realizou uma série de ações intermediárias para que a tarefa fosse concluída com êxito.

No entanto, se a mesma "instrução" fosse dada a uma máquina (e ela não tivesse qualquer outra orientação prévia armazenada), ela não conseguiria "trazer a pasta desejada".

Para a máquina, é necessário que a "instrução" seja detalhada em pequenas etapas, visto que ela é construída para ser capaz de entender só dessa forma, ou seja, em pequenas operações.

No exemplo em questão, a máquina deveria receber um conjunto de instruções como o da Fig. 6.17, específicas para ela, sendo, portanto, chamadas de instruções de máquina (da máquina).

O programa mostrado na Fig. 6.17 — não completo nem estruturado — tem apenas o propósito de indicar, por comparação simbólica, a diferença entre a generalidade de uma instrução para o funcionário e o detalhe de uma instrução para a máquina. Com os sistemas de computação isto não é diferente.

Uma instrução de máquina é a formalização de uma operação básica (ou primitiva) que o hardware é capaz de realizar diretamente. Podemos, por exemplo, fabricar um processador com uma UAL capaz de somar ou de multiplicar dois números, mas ainda não se fabricou uma UAL capaz de executar:

$X = A + B*C$

```
1  • achar arquivo. Se não houver arquivo, vá p/3; senão, prosseguir
   • comparar nº arquivo com nº arquivo que contém as pastas dos funcionários
   • se números iguais, então prosseguir; senão, voltar para 1
2  • achar uma pasta. Se não houver mais pastas, vá p/3; senão, prosseguir
   • comparar nome da pasta com nome dado
   • se forem iguais, então prosseguir; senão, voltar para 2
   • retirar a pasta
   • abrir a pasta para quem pedir
3  • parar
```

Figura 6.17 Exemplo de instruções primitivas ou instruções para uma máquina localizar e buscar uma pasta de documentos de um arquivo.

de uma só vez. A UAL tem que ser instruída para executar, em primeiro lugar:

$T = B*C$

Em seguida, ela realizará a operação:

$X = A + T$

Primeiramente, a UAL efetuará a multiplicação, cujo resultado é temporariamente armazenado em algum tipo de memória (dependendo do programa e do sistema), que poderia ser um registrador ou uma célula de memória, para, em seguida, este resultado parcial ser recuperado e somado ao valor A.

O projeto de um processador é centrado no conjunto de instruções de máquina que se deseja que ele execute (na realidade, do conjunto de operações primitivas que ele poderá executar). Uma das mais fundamentais análises e decisões do projeto envolve o tamanho e a complexidade do conjunto de instruções. Quanto menor e mais simples o conjunto de instruções, mais rápido é o ciclo de tempo do processador.

Atualmente, há duas tecnologias de projeto de processadores empregadas pelos fabricantes de mini, microcomputadores e de estações de trabalho:

- *Sistemas com conjunto de instruções complexo* (complex instruction set computers — CISC) e
- *Sistemas com conjunto de instruções reduzido* (reduced instruction set computers — RISC).

Neste capítulo, trataremos basicamente dos sistemas que se utilizam de tecnologia CISC (como todos os microprocessadores do tipo PC ou Macintosh), deixando para o Cap. 11 a descrição dos sistemas RISC e uma comparação entre essas tecnologias.

Na realidade, todo o projeto do processador se resume em:

- definir o conjunto de instruções (qual o formato e tamanho de cada uma, quais as operações a realizar — o que caracteriza a quantidade);
- implementar os componentes do processador em função da definição anterior (UAL, registradores, barramentos, etc.).

Do ponto de vista físico (do ponto de vista do hardware), uma instrução de máquina é um grupo de bits que indica ao processador uma operação ou ação que ele deve realizar. Um processador é fabricado com a capacidade de realizar uma certa quantidade de operações bem simples (primitivas), cada uma delas associada a uma instrução de máquina.

Funcionalmente, um processador possui instruções capazes de realizar os seguintes tipos de operação:

- operações matemáticas (aritméticas, lógicas, de complemento, de deslocamento);
- movimentação de dados (memória — UCP e vice-versa);
- entrada e saída (leitura e escrita em dispositivo de E/S); e
- controle (desvio da seqüência de execução, parar, etc.).

Quando se escreve "conjunto de instruções", estamos nos referindo a todas as possíveis instruções que podem ser interpretadas e executadas por um processador. O processador Intel 8080 possuía um conjunto de 78 instruções de máquina, enquanto o Intel 8088 possuía 117 instruções, o 80486 possuía um conjunto com 286 instruções de máquina e o Intel Pentium II tem seu conjunto de instruções com 217 delas.

NOTA: A partir deste ponto, vamos simplificar a nomenclatura e o entendimento do assunto instruções. O termo *instrução* será usado quando estivermos nos referindo à instrução de máquina, como instrução binária ou Assembly (ver item 6.5), e o termo *comando* quando nos referirmos a uma instrução de linguagem de mais alto nível, como Pascal, C, Delphi, Lisp, Fortran, etc.

6.3.2 Formato das Instruções

De modo geral, podemos separar o grupo de bits que constitui a instrução em duas partes: uma delas indica o que é a instrução e como será executada e a outra parte se refere ao(s) dado(s) que será(ão) manipulado(s) na operação. A primeira parte é constituída de um só campo, enquanto a segunda parte poderá ter um ou mais campos, conforme a instrução se refira explicitamente a um ou mais dados. Assim, temos os seguintes campos em cada instrução:

- um campo (um subgrupo de bits) chama-se *código de operação*;
- o restante grupo de bits (se houver) denomina-se *campo do(s) operando(s)* ou, simplesmente, operando(s).

Código de operação — *C.Op.* — é o campo da instrução cujo valor binário é a identificação (ou código) da operação a ser realizada. Assim, cada instrução possui um único código, o qual servirá de entrada no decodificador da área de controle (ver item 6.2.2.5). A Fig. 6.18 apresenta exemplos de tipos de operações primitivas normalmente encontradas na implementação dos processadores.

Um processador que possua instruções cujo campo C.Op. tenha uma largura de 8 bits poderá ser fabricado contendo a implementação de um conjunto de até 256 instruções diferentes, visto que:

C.Op. = 8 bits. Então: $2^8 = 256$ códigos de operação.

Como cada C.Op. representa uma única instrução, então 256 C.Op. indica 256 instruções de máquina.

Campo operando — *Op.* — é(são) o(s) campo(s) da instrução cujo valor binário indica a localização do dado (ou dados) que será(ão) manipulado(s) durante a realização da operação.

*** Transferir uma palavra de dados de uma célula para outra.**
*** Efetuar a soma entre dois operandos, guardando o resultado em um deles ou em um terceiro operando.**
*** Desviar incondicionalmente para outro endereço fora da seqüência.**
*** Testar uma condição. Se teste verdadeiro, então desviar para outro endereço fora da seqüência.**

*** Realizar uma operação lógica AND entre dois valores.**
*** Parar a execução de um programa.**
*** Adicionar 1 ao valor de um operando.**

*** Transferir um byte de dados de uma porta de E/S para a MP.**
*** Transferir um byte de dados da MP para uma porta de E/S.**
*** Substituir o operando por seu valor absoluto.**

Figura 6.18 Exemplo de operações primitivas típicas.

Como já mencionado antes, a instrução pode ser constituída de um ou mais campos "operando", isto é, se a operação for realizada com mais de um dado, a instrução poderá conter o endereço de localização de cada um dos dados referidos nela. Por exemplo, uma instrução que defina uma operação de adição de dois valores pode indicar explicitamente o endereço de cada um dos dois valores (operando 1 e operando 2), bem como o endereço no qual o resultado será armazenado (operando 3). Nesse caso, o seu formato seria:

C.Op.	**Operando 1**	**Operando 2**	**Operando 3**

Utilizando-se uma forma mais específica para representar a operação que a instrução indica, teríamos:

(operando 3) ← (operando 1) + (operando 2)

A mesma operação (necessitando da mesma forma de três operandos) poderia ser realizada com outro tipo de instrução:

C.Op.	**Operando 1**	**Operando 2**

Neste último exemplo, a instrução pode indicar (pelo C.Op. específico e diferente do C.Op. da instrução anterior) que se deve somar o valor indicado pelo operando 1 com o valor indicado pelo operando 2, e que o resultado deve ser armazenado no operando 1 (poderia também ser no operando 2).

A instrução seria assim representada:

(operando 1) ← (operando 1) + (operando 2)

Ou ainda:

(operando 2) ← (operando 1) + (operando 2)

Também poderíamos utilizar o registrador acumulador (ACC) para armazenar inicialmente um dos valores e, posteriormente, depois da realização da soma, armazenar o resultado. Como em processadores dotados de ACC, ele tende a ser um só, não há necessidade de explicitar seu endereço; basta programar a UC para utilizá-lo quando decodificar o C.Op. específico.

A instrução teria o seguinte formato:

C.Op.	Operando

E sua representação:

ACC ← ACC + (Operando)

Pode-se observar, então, que, em um mesmo conjunto de instruções de um processador, podem existir formatos diferentes de instruções, inclusive para a realização de uma mesma operação. Os formatos apresentados são uma parte da quantidade de formatos que podem existir distribuídos em processadores reais.

Uma análise mais detalhada sobre os tipos de instrução que podem ser criados e utilizados, bem como uma comparação entre os fatores positivos e negativos de cada caso, será efetuada no Cap. 8, quando se trata especificamente do formato e funcionamento das instruções de máquina dos processadores.

Um outro fator a ser considerado no projeto do conjunto de instruções de um processador refere-se ao significado do valor binário indicado no(s) campo(s) operando(s) das instruções. Ou seja, o modo de localizar o dado pode variar de instrução para instrução. Chama-se a isto de **modo de endereçamento**; atualmente há vários destes modos sendo empregados nos processadores. Para evitar o risco de tornar a explicação inicial do assunto mais complicada do que o desejado, deixaremos, como no caso anterior, mais detalhes para o Cap. 8. Neste capítulo apresentaremos apenas instruções com formato e características bem simples: apenas dois modos de endereçamento e somente 1 operando.

6.3.3 Considerações sobre o Formato das Instruções

No que se refere à definição do código de operação, C.Op., há duas maneiras de se criar um conjunto de instruções de um processador:

- instruções com C.Op. de tamanho fixo; e
- instruções com C.Op. de tamanho variável.

Conjuntos de instruções com C.Op. de tamanho fixo são mais simples de implementar e de manipular durante a execução de um programa. Porém, em sistemas que possuem uma grande quantidade de instruções, o tamanho do C.Op. tem de crescer o suficiente para acomodar todos os códigos necessários; com isso, aumenta o tamanho das instruções e, conseqüentemente, o tamanho requerido pelo programa na MP, o que é, de um modo geral, uma desvantagem.

Neste tipo de C.Op. pode-se calcular imediatamente a quantidade máxima de instruções que pode ser implementada no respectivo processador, apenas sabendo-se a quantidade de bits do campo.

Por exemplo, um conjunto de instruções que tenham C.Op. de 6 bits pode ter, no máximo, 64 códigos diferentes, ou seja, 64 instruções podem ser criadas.

Nunca se deve esquecer que a memória, apesar dos avanços tecnológicos que têm reduzido o seu custo e da maior capacidade por pastilha, ainda representa uma parcela razoável no preço total de um sistema de computação, não devendo, portanto, ser desperdiçada. Um valor típico de C.Op. de tamanho fixo é 8 bits.

Os microprocessadores Intel 8080 e 8085 são exemplos de utilização de código de operação (C.Op.) de tamanho fixo igual a 8 bits. Embora seja possível configurar 256 diferentes códigos com 8 bits, este não foi o caso dos dois processadores referidos (o 8080 possuía um conjunto de 78 instruções; e o 8085, um conjunto de 80 instruções), visto que, na realidade, cada código variava os 6 bits mais significativos, deixando os 2 bits mais à direita para indicações de especificidade na instrução.

Instruções que possuem C.Op. de tamanho variável permitem codificar uma quantidade maior de instruções com menor quantidade de bits, embora muitas vezes se personalize o tamanho do campo operando (reduzindo-se, com isso, a quantidade de endereçamento de memória), ou se tenha de aumentar o tamanho total da instrução, acarretando os prejuízos de gasto de memória já mencionados.

Basicamente, o C.Op. de tamanho variável permite que se estabeleça um compromisso mais versátil entre a quantidade de bits do código de operação e a quantidade de bits do(s) campo(s) operando(s), de modo a se criar um conjunto de instruções que atenda ao requisito de mais instruções, com quantidades diferentes de operandos, sem aumentar demasiadamente o tamanho total das instruções.

Nos processadores Intel 8086, 8088 e 80286 o código de operação tinha 1 byte de tamanho, enquanto nos processadores Intel 386, 486 e Pentium, o código de operação pode ter 1 ou 2 bytes de tamanho. Os processadores Motorola, da família 68000, possuem instruções com código de operação de tamanho variável entre 1/2 e 2 bytes.

O mais comum é utilizar um conjunto de instruções com C.Op. de tamanho variável, com instruções de tamanho total também variável.

6.4 FUNCIONAMENTO DA UCP. O CICLO DA INSTRUÇÃO

A base do projeto de uma UCP é a escolha do conjunto de instruções que ela irá executar (trata-se de definir que operações o hardware será capaz de realizar diretamente através de seus circuitos), para em seguida definir e especificar os demais componentes da arquitetura e da organização, os quais contribuirão para o processo de interpretar e executar cada instrução.

Neste item será mostrado, em detalhe, o funcionamento básico de um processador ao descrevermos, passo a passo, as etapas requeridas para o processador completar a execução de uma instrução de máquina. Em seguida, serão apresentadas algumas considerações sobre o conjunto de instruções dos processadores reais.

Para descrever com clareza as referidas etapas e permitir o entendimento do sentido individual e global do processo de funcionamento da UCP, vamos utilizar um subsistema UCP/MP hipotético, mais simples que os modelos comerciais. A Fig. 6.19 relaciona as características básicas do subsistema UCP/MP hipotético, o qual será utilizado no decorrer das explicações deste item. As instruções JZ Op., JP Op., JN Op., JMP Op., mostradas nessa figura, denominam-se instruções de desvio. *Desvio* é uma alteração forçada da seqüência de execução de um programa. (No Cap. 9 será apresentada uma explicação mais detalhada sobre instruções de desvio e suas conseqüências na execução de um programa.)

A Fig. 6.20 descreve, com mais detalhe, o fluxo de ações que constituem um ciclo de instrução, ampliando o fluxo mostrado na Fig. 6.2.

O ciclo de instrução apresentado pelo fluxo da Fig. 6.20 pode ser descrito em LTR (linguagem de transferência entre registradores — ver item 5.3.3), de modo que possamos acompanhar sua realização com a movimentação de informações nos componentes da UCP/MP da Fig. 6.19:

Iniciar	
RI ←(CI)	buscar a instrução, cujo endereço está no CI
(CI) ← (CI) + 1	conteúdo de CI é incrementado para o endereço da próxima instrução
Interpretar o C.Op.	o decodificador recebe os bits do C.Op. e gera uma saída para a UC
Buscar Op. (se houver)	
Executar a instrução	
Retornar	

Não vamos, neste momento, considerar alguns aspectos, como a possibilidade de ocorrer a detecção de um sinal de interrupção (fenômeno que será discutido no Cap. 10) ou a execução de vários ciclos de instrução de modo paralelo (método de *pipelining*, que será abordado no item 6.6), ainda com o mesmo propósito de simplificar o modelo, facilitando a explicação.

Características de um processador simples a ser utilizado nos exemplos

1 - Palavra: 12 bits

2 - Endereços: 8 bits (256 células de memória)

3 - Células de 12 bits

4 - Instruções de 1 operando apenas, com C. Op. = 4 bits e campo operando = 8 bits

5 - Campo operando sempre indica o end. de memória do dado, exceto em instruções de desvio

6 - A UCP possui apenas um registrador de dados, o ACC, com 12 bits de tamanho, o RI, também com 12 bits de tamanho, o CI e o REM com 8 bits cada um e o RDM com 12 bits também.

7 - Formato das instruções

C. Op.	Operando
4 bits	8 bits

8 - Instruções disponíveis:

C. Op.	Sigla	Descrição
0	HLT	Parar a execução do programa*** Halt, em inglês
1	LDA Op.	ACC ← (Op.)*** Load, em inglês
2	STR Op.	(Op) ← ACC*** Store
3	ADD Op.	ACC ← ACC + (Op.)
4	SUB Op.	ACC ← ACC - (Op.)
5	JZ Op.	Se ACC = 0, então: CI ← Op.
6	JP Op.	Se ACC > 0, então: CI ← Op.
7	JN Op.	Se ACC < 0, então: CI ← Op.
8	JMP Op.	CI ← Op.
9	GET Op.	Ler dado da porta de entrada e armazená-lo em (Op.)
A	PRT Op.	Colocar na porta referente à impressora o valor armazenado em (Op.).

Figura 6.19 Descrição das características principais de um pequeno processador.

Para que a explicação seja mais interessante, vamos considerar a execução de duas das instruções definidas no modelo UCP/MP da Fig. 6.19:

LDA Op e **ADD Op**

e, utilizando a organização de MP e UCP dessa mesma figura, vamos considerar alguns valores iniciais (a Fig. 6.21 mostra os detalhes) existentes ao iniciar a execução do primeiro ciclo de instrução — da *instrução LDA:*

a) A instrução LDA está armazenada na MP no endereço decimal 2, que é igual a 02_{16} e a 00000010_2. Sua descrição em binário é: 000110110100, ou 1B4 em hexadecimal.

Os 12 bits que constituem a instrução têm finalidades diferentes conforme o formato já definido para as instruções (Fig. 6.19), sendo os 4 primeiros para o código da operação (C.Op.) e os 8 restantes para indicar o valor do campo operando, endereço do dado a ser manipulado pela operação (Op.).

Assim, temos: C.Op. 1_{16} (0001_2) e Op. = $B4_{16}$ (10110100_2) (valor escolhido no exemplo).

0001	10110100

b) O valor do dado armazenado na célula de endereço B4 é igual a 423_{10} ou $1A7_{16}$ (trata-se de um valor assumido para o exemplo).

c) A instrução ADD está armazenada no endereço 03_{16}, e sua descrição em binário é: 001110110101 ou $3B5_{16}$.

Como na instrução anterior, os 4 primeiros bits constituem o C.Op. e os 8 restantes o valor de Op.

0011	10110101
C.Op.	Op.

sendo C.Op. = 3_{16} (conforme definido no exemplo da Fig. 6.19 para a instrução ADD) e Op. = $B5_{16}$ (valor também assumido para o exemplo).

d) O valor do dado armazenado em $B5_{16}$ é $07D_{16}$.

e) O valor armazenado no CI = 02_{16} (este valor é considerado no exemplo como tendo sido atribuído pelo sistema operacional).

f) O valor armazenado no RI = 317_{16} (provavelmente é o valor da instrução anteriormente executada).

g) O valor armazenado no ACC = $20B_{16}$ (também é um valor obtido em operação anterior).

h) O valor armazenado no REM = B3 e no RDM = 7BC.

Então, ao terminar um ciclo de uma instrução qualquer, a UC reinicia o processo através da execução do ciclo de uma nova instrução (no caso, será a instrução LDA de nosso exemplo), conforme o fluxo da Fig. 6.20.

Seguindo as etapas indicadas na referida figura e observando os valores armazenados e alterados nas Figs. 6.22 (subciclo de busca) e 6.23 (subciclo de execução), teremos (todos os valores estão representados em hexadecimal):

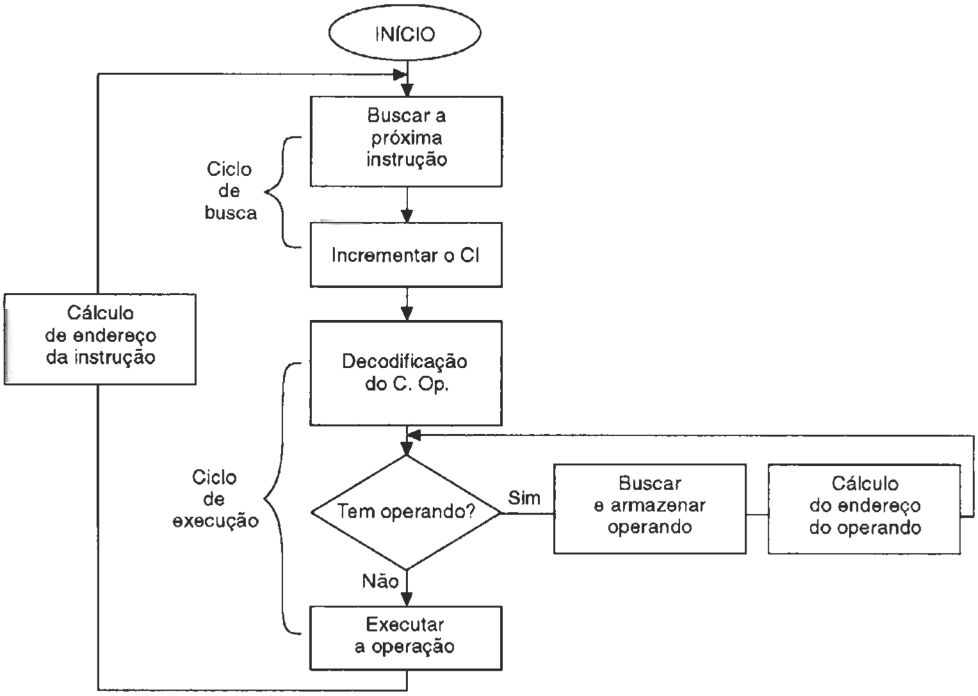

INÍCIO
Buscar a próxima instrução
Ciclo de busca
Incrementar o CI
Cálculo de endereço da instrução
Decodificação do C. Op.
Ciclo de execução
Tem operando?
Sim
Buscar e armazenar operando
Cálculo do endereço do operando
Não
Executar a operação

Figura 6.20 Fluxograma de um ciclo de instrução.

1 - RI ← (CI);

Resultado: RI = 1B4 (02) (Fig. 6.22)

Descrição passo a passo:

a) A UC aciona a transferência dos bits do CI para o REM (cópia), pelo barramento interno.

Resultado: CI → REM = 02 (Fig. 6.22)

b) A UC ativa a linha READ (leitura) do barramento de controle, o qual é reconhecido pelo circuito decodificador de endereços da MP.

c) O dispositivo de controle da memória decodifica o endereço. Em seguida, aquele circuito transfere os bits (cópia) da célula de endereço 02, cujo valor é 1B4 para o barramento de dados e daí para o RDM.

Resultado RDM = 1B4 (Fig. 6.22)

d) No instante seguinte, o valor 1B4 é transferido do RDM para o RI, pelo barramento interno do processador.

NOTAS: 1. Observe a notação usada, (RI) e (CI), ou seja, RI e CI entre parênteses, para indicar conteúdo, porque (02) significa "trazer o conteúdo do endereço 02". Se fosse usado o termo CI apenas (sem parênteses), o próprio valor 02 seria transferido para o RI.

2. Para simplificar a explicação, foram omitidos alguns eventos que ocorrem na realidade de um ciclo destes, como por exemplo, a colocação, pelo circuito de controle da memória, de um sinal no barramento de controle confirmando o término da transferência do valor no barramento de dados, de modo que o processador possa utilizá-lo (transferência para o RDM, etc.).

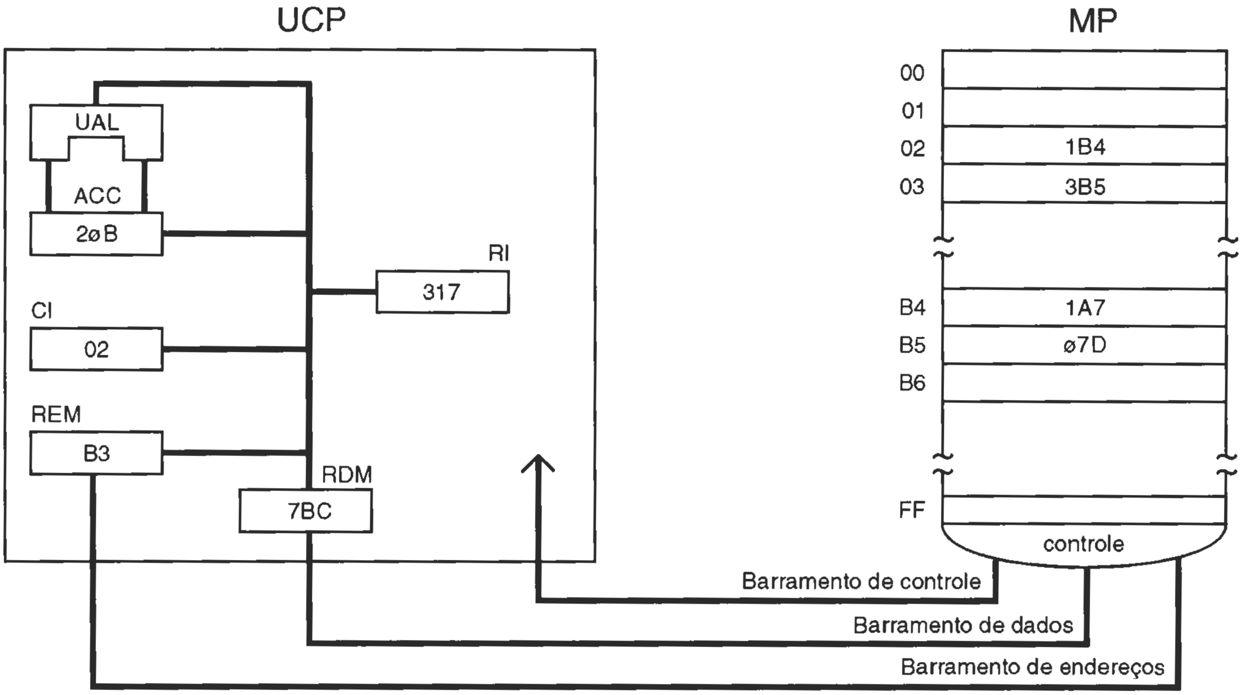


Figura 6.21 Dados cruciais para execução do ciclo das instruções LDA e ADD.

2 - CI ← CI + 1;

Resultado: CI = 03 (Fig. 6.22)

Como no exemplo adotado cada instrução ocupa uma célula da MP (especificação da Fig. 6.19), e as instruções estão organizadas em seqüência, a próxima instrução deverá ocupar a célula seguinte, cujo endereço será, então, 03. Portanto, 02 + 01 = 03, que é o endereço seguinte.

No entanto, em quase todos os sistemas em funcionamento não há processadores tão "bem-comportados" quanto o especificado na Fig. 6.19, especialmente no que se refere à relação entre o tamanho das instruções e das células de MP. Por conseguinte, a realidade está mais para:

(CI) ← (CI) + n, sendo **n** = quantidade de células ocupadas por uma única instrução, em vez de: (CI) ← (CI) + 1

Além disso, em sistemas (e não são poucos hoje em dia) em que o tamanho das instruções é variável, o valor de **n** é variável também e a UC deve ser preparada para este fato.

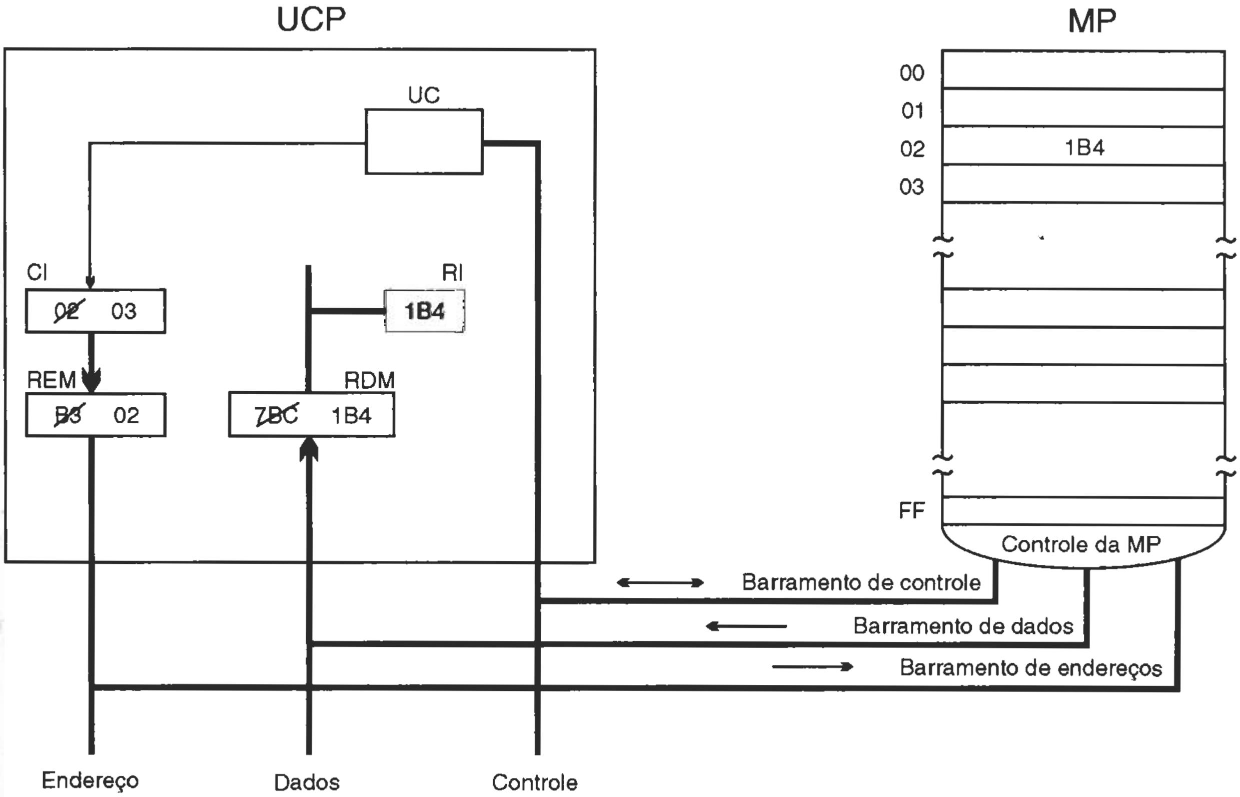


Figura 6.22 Fluxo de dados e de endereços durante a realização do ciclo de busca da instrução LDA.

Nos microprocessadores 8080/8085, cujas instruções podiam ocupar 1, 2 ou 3 células de memória (cada célula tinha 8 bits de largura), as instruções eram lidas para a UCP um byte de cada vez (tamanho da célula) e o CI era incrementado de 1 em 1, porém mais de uma vez durante o ciclo da mesma instrução (se a instrução ocupasse 2 ou 3 bytes). Nos sistemas IBM/370, os 2 primeiros bits do código de operação indicavam o tamanho da instrução, sendo 00 para instruções de 2 bytes, 01 para instruções de 4 bytes e 11 para instruções de 6 bytes de tamanho, de modo que o CI era incrementado de acordo.

OBS.: Para um melhor entendimento do fluxo de dados nos exemplos, se ocorrer uma operação de escrita em célula de MP ou registrador, a figura mostrará os dois valores: o anterior à esquerda, com uma barra diagonal atravessada; e o novo valor à direita.

3 - Decodificação do código de operação — Fig. 6.23

a) A UC emite o sinal apropriado para que o RI transfira para o decodificador de instrução os 4 bits mais significativos que correspondem ao valor do C.Op.

Resultado: decodificador ← 0001_2 ou 1_{16} (Fig. 6.23)

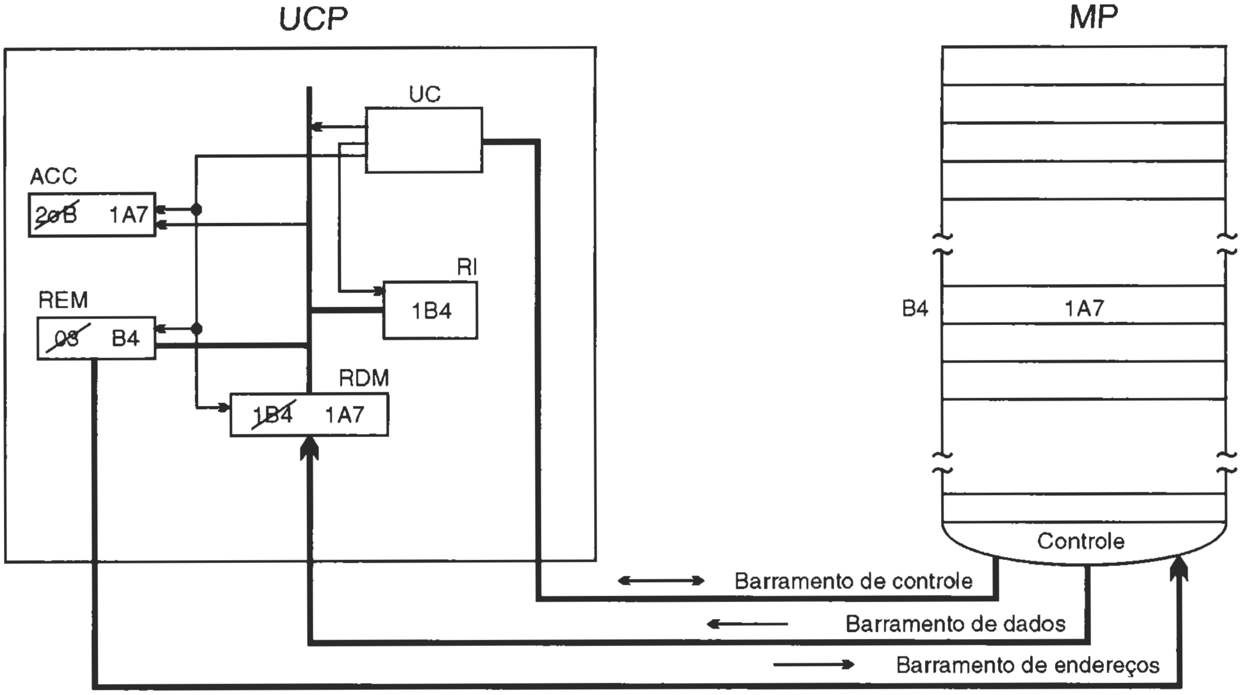


Figura 6.23 Fluxo de dados e de endereços durante o ciclo de execução da instrução LDA.

b) O decodificador seleciona, através da lógica nele existente (ver exemplos de decodificadores no Cap. 2 e no item 5.7), a linha de saída correspondente para a UC a qual emitirá os sinais adequados e na seqüência preestabelecida que conduzirão a execução da operação definida pela instrução.

Na realidade, a ação decorrente da saída decodificada do C. Op. depende do método utilizado pelo sistema para executar as instruções, seja por microprogramação seja por programação diretamente no hardware.

4 — Se tiver operandos, buscá-lo(s); senão, passar para o item 5 (Fig. 6.23)

No presente caso, não há operando a ser previamente buscado.

5 — Execução da operação

Resultado: ACC = 1A7 (Fig. 6.23).

a) A UC emite o sinal para que os bits correspondentes ao valor do campo do operando da instrução B4 sejam transferidos para o REM, pelo barramento interno.

Na prática, em geral há um cálculo para se achar o valor de um endereço, como acontecia nos microprocessadores de 16 bits e ainda acontece nos processadores de 32 bits atuais (sejam Intel ou Motorola). No nosso sistema "bem-comportado" isto não é necessário, e o endereço B4 é transferido diretamente para o REM.

b) A UC ativa a linha READ (leitura) do barramento de controle, o qual aciona o circuito de controle da MP para decodificar o endereço B4.

c) Decodificado o endereço, o circuito de controle da MP transfere o valor (o conteúdo) armazenado na célula de endereço B4 — cujo valor é 1A7 — para o RDM, pelo barramento de dados.

Resultado: RDM = 1A7

d) No instante seguinte, a UC emite o sinal apropriado para que este valor seja transferido (cópia) para o ACC pelo barramento interno do processador.

Os passos 1 e 2 correspondem ao ciclo *de busca* (*fetch*), e os seguintes — 3, 4 e 5 — correspondem ao ciclo *de execução*. O ciclo de instrução realizou dois acessos à memória para realização de dois ciclos de leitura.

O que, na prática, realmente diferencia o desempenho de uma instrução em relação à outra é a quantidade de ciclos de memória (acessos) que cada uma realiza durante seu ciclo de instrução, visto que o ciclo de memória é um tempo ponderável se comparado com o ciclo do processador.

Vamos, em seguida, descrever o **ciclo da instrução ADD Op.**, considerando que é a instrução seguinte na seqüência da execução. (A Fig. 6.17 mostra a instrução no endereço 03 da MP, justamente o valor atualmente armazenado no CI, após a execução do ciclo de LDA Op. — ver Fig. 6.23.)

A Fig. 6.24 mostra o subciclo de busca de ADD Op., e a Fig. 6.25 mostra o subciclo da execução. Nesta instrução será usado o passo 4 do ciclo de instrução (buscar operando) e haverá efetivamente a realização de uma operação (operação aritmética de adição), com a conseqüente ação da UAL.

Dados a serem inicialmente considerados, referentes ao término da instrução anterior:

CI = 03; RI = 1B4; ACC = 1A7; REM = B4; RDM = 1A7

Seguindo as etapas indicadas na Fig. 6.20, teremos:

1 - RI ← (CI)

a) REM ← CI = 03 (Fig. 6.24)

b) Ativação da linha READ pela UC. Decodificação pelo circuito de controle da MP do endereço colocado no barramento de endereços.

c) RDM ← 3B5. Naturalmente, o valor copiado da célula (3B5) vai primeiro para o barramento de dados e daí para o RDM, conforme já explicado no exemplo anterior.

d) RI← 3B5.

2 - (CI) ← (CI) + 1

CI = 03 + 1 = 04 (Fig. 6.24)

3 - Decodificação do código de operação

C.Op. = 3 (Fig. 6.24)

a) Decodificador ← (RI(C.Op.))

A definição da saída decodificada correspondente à operação ADD é enviada para a UC.

b) A UC emite sinais apropriados para a realização dos passos 4 e 5, de acordo com sua programação prévia para esse código de operação.

4 - Buscar operando na MP (Fig. 6.25)

Como a instrução determina que o valor armazenado no ACC (1A7) seja somado a um valor que está na MP, no endereço B5, este valor (operando) deve ser transferido da MP para a UAL (na realidade é transferida uma cópia do valor, permanecendo este também na sua célula) de modo que, em seguida, possa ser somado. Trata-se, então, de realizar um ciclo de memória para leitura.

a) A UC emite sinal de controle de modo que:

REM ← Op. REM ← B5, pois Op. = B5

b) O valor B5 é colocado no barramento de endereços (UC) e a UC ativa a linha READ.

c) O controle da MP decodifica o endereço B5 e, em seguida, os bits armazenados no endereço B5 (07D) são transferidos para o RDM.

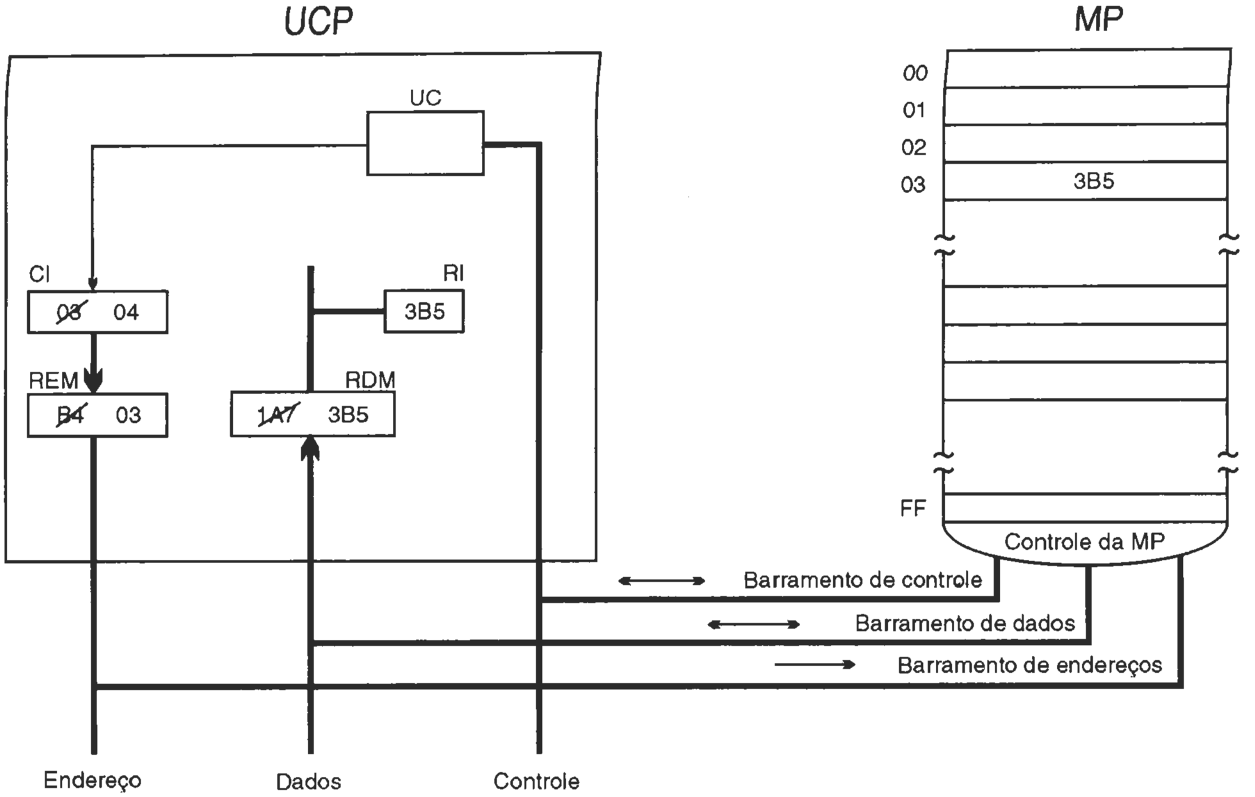

UCP
MP
UC
CI
~~03~~ 04
RI
3B5
REM
~~B4~~ 03
RDM
~~1A7~~ 3B5
00
01
02
03
3B5
FF
Controle da MP
Barramento de controle
Barramento de dados
Barramento de endereços
Endereço
Dados
Controle

Figura 6.24 Fluxo de dados e de endereços durante a realização do ciclo de busca da instrução ADD.

RDM ← (M(REM))
RDM ← (M(B5))
RDM = 07D

d) UAL ← ACC (1A7), que é o primeiro operando

ACC ← RDM (07D)
UAL ← ACC (07D), que é o segundo operando

Há diversos modos de implementar a colocação de valores na UAL. Certos sistemas usam registradores temporários para armazenar os dois operadores imediatamente antes de serem transferidos para a UAL. Em outros, o barramento interno leva dados diretamente para a UAL, quando desejado.

No nosso sistema hipotético, vamos utilizar o ACC como meio de ligação entre o barramento interno e a UAL.

5 - Execução da operação (Fig. 6.25)

Nesta etapa, a UC emite o sinal correspondente que aciona a entrada dos dois valores no circuito lógico, que realiza uma adição e se obtém o resultado na saída.

1A7 + 07D = 224

A soma foi realizada usando-se aritmética hexadecimal para números inteiros sem sinal (ver Cap. 3).

O resultado, valor 224, é transferido para o ACC, concluindo a execução do subciclo de execução e o ciclo completo da instrução.

Se o programa continuasse em execução, neste momento seria iniciado um outro ciclo de instrução, correspondente ao valor armazenado no endereço 04.

A instrução ADD consumiu dois ciclos de memória (para leitura), um para a busca da instrução e outro para a busca do 2.° operando.

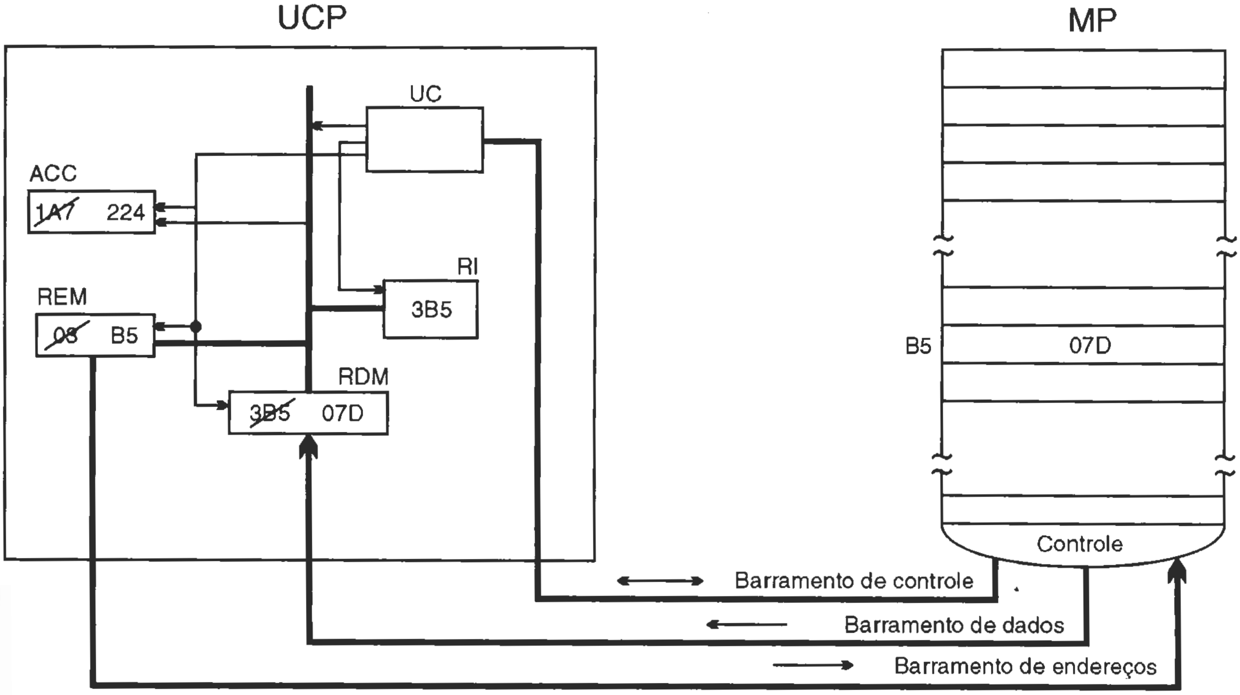


Figura 6.25 Fluxo de dados e de endereços durante a realização do ciclo de execução da instrução ADD.

Em termos comparativos, seu tempo básico de execução é praticamente o mesmo da instrução LDA, visto que ambas realizaram a mesma quantidade de acessos à MP, ou seja, dois acessos. A pequena diferença existente reside na operação aritmética realizada pela instrução ADD. No entanto, se a instrução a ser executada possuísse dois operandos, sendo, por exemplo, do tipo:

$(\text{Op.}_1) \leftarrow (\text{Op.}_1) + (\text{Op.}_2)$

então, a completa realização de seu ciclo de instrução consumiria 4 ciclos de memória (4 acessos), o dobro das duas instruções exemplificadas: um ciclo de memória para buscar a instrução e armazená-la no RI; um ciclo de memória para buscar o primeiro operando (Op.$_1$); um terceiro ciclo para buscar o segundo operando e, finalmente, o último ciclo de memória para armazenar o resultado da operação na memória, no endereço indicado por Op.$_2$.

Ao mencionar ciclo de memória, estamos nos referindo ao conjunto de etapas que leva a concluir um ciclo de leitura ou um ciclo de escrita, e não aos ciclos de relógio que efetivamente acionam a efetivação de uma microoperação (um dos passos de uma das citadas operações de memória). Na realidade, a execução de um ciclo de instrução consome vários pulsos de relógio, variando-se sua quantidade de acordo com o tipo da instrução.

6.5 LINGUAGEM DE MONTAGEM (ASSEMBLY)

No item anterior mostramos a execução de duas instruções, definindo para ambas valores numéricos de código de operação e de dados. De modo que fomos descrevendo cada passo do ciclo de instrução através da visão do fluxo dos valores binários (para maior simplicidade, utilizamos valores hexadecimais para substituir o valor real em binário). Se fôssemos utilizar este procedimento (valores em hexadecimais para instruções, códigos de operação, dados), e se considerássemos programas extensos, o processo seria extremamente cansativo e complicado.

Realmente, a linguagem de máquina (linguagem binária de 0s e 1s) é a mais elementar, a mais direta forma de utilizar o hardware. Mas é também a mais complicada e difícil para o programador.

Vamos exemplificar através da descrição de um simples comando em linguagem Pascal e de um programa equivalente em linguagem de máquina, mostrado na Fig. 6.26, usando-se as instruções definidas na Fig. 6.19.

Consideremos o seguinte comando escrito em linguagem Pascal:

```
X: = A + B - C;
```

(a) Comando em Pascal

```
000100100011          123
001100100100          324
010000100101          425
001000100110          226
```

(b) Programa em linguagem de máquina equivalente ao comando em (a)

(c) Programa em hexadecimal equivalente ao programa em (b)

```
PROG  SOMA
LDA   A
ADD   B
SUB   C
STR   X
```

(d) Programa em Assembly equivalente ao comando em (a)

Figura 6.26 Exemplo de modos diferentes (e trabalhosos) de escrever um programa, equivalente a um único comando em uma linguagem de alto nível, como Pascal.

A Fig. 6.26(b) mostra o comando convertido em linguagem de máquina. Se para um conjunto bastante pequeno de quatro instruções, usando-se palavras de apenas 12 bits, já é trabalhoso e com grande probabilidade de erro, imagine-se o caso de um programa com centenas de linhas de código em um sistema com palavras de 32 bits. Podem ser páginas e páginas com números absolutamente ininteligíveis.

Uma ligeira melhoria de apresentação para o programador (se é que hoje em dia existe algum programador que utilize este tipo de instruções) consiste em substituir os valores binários por números hexadecimais. Apesar de se continuar trabalhando com números, reduz-se consideravelmente a quantidade de algarismos (divide-se por 4), e a apresentação dos elementos se torna um pouco menos complicada (muito pouca melhora mesmo). A Fig. 6.26(c) mostra o programa anterior, porém representado em linguagem hexadecimal. Este, aliás, é o método freqüentemente utilizado em todos os manuais e livros: quando se trata de representar valores binários internos da máquina, usa-se o hexadecimal equivalente.

A primeira evolução para tornar os programas mais representativos da intenção do programador e, portanto, mais inteligíveis ao próprio, foi com o emprego de símbolos alfanuméricos em vez de números. O ser humano está mais acostumado com o significado de uma ação ser expresso de forma mais direta e de acordo com os símbolos que ele aprendeu a usar.

Para o programador que precise usar uma instrução de adição, é mais simples entendê-la se estiver expressa como ADD (palavra inglesa cujo significado é soma) do que pelo número 3 (código de operação de ADD no sistema hipotético que criamos na Fig. 6.19).

Desenvolveu-se, então, uma linguagem de símbolos (e não de números) alfabéticos, denominada *linguagem de montagem* (em inglês chama-se Assembly). A Fig. 6.26(d) mostra o programa equivalente ao comando Pascal da Fig. 6.26(a), escrito em linguagem de montagem.

O programa, como o das Figs. 6.26(b) e 6.26(c), compreende uma linha de código para cada instrução, tendo, portanto, uma relação de 1:1 com as instruções de máquina, porém possuindo muito mais facilidade de compreensão e manuseio do que as instruções binárias ou mesmo hexadecimais.

Em geral, uma linha de instrução Assembly é composta de quatro partes ou campos (ver fig. 6.27):

1) *Rótulo* (ou label) — indica um endereço significativo no programa, como, por exemplo, o endereço de início do programa (sempre é colocado para o sistema se orientar sobre o início), o endereço de desvio em um *loop* e outros. No exemplo da Fig. 6.26(d), PROG, A, B, C e X são rótulos. As demais linhas não usam rótulos, o que significa que o endereço da linha é o seguinte na seqüência.

2) *Operação* — contém o mnemônico predefinido adequadamente para simbolizar a correspondente operação. Conforme já mencionado, ADD, SUB, LDA e STR simbolizam melhor a operação desejada do que 3, 4, 1 e 2.

3) *Operando(s)* — onde são inseridos os símbolos representativos dos endereços de memória ou dos registradores utilizados pela instrução para armazenar o(s) respectivo(s) dado(s) referido(s) pela instrução. No exemplo da Fig. 6.26(d), os dados estão na MP e, por isso, A, B, C e X representam endereços de memória.

4) *Comentários* — é um campo opcional, ignorado durante o processamento do programa. Serve apenas para auxiliar o entendimento do programa, de modo idêntico ao que se faz em programas escritos em linguagem de mais alto nível.

Programas escritos em linguagem de montagem (Assembly) não são diretamente executáveis porque o hardware não entende símbolos, mas sim códigos de operação binários, etc. Desta forma, há necessidade de se desenvolver um programa com o propósito de converter as instruções "Assembly" em correspondentes instruções de máquina, e os endereços simbólicos (A, B, C, X do exemplo) em endereços físicos de memória. Este programa denomina-se montador ("Assembler") e é específico da UCP em que ele será executado. Isto fica claro quando se sabe que o montador cria instruções de máquina, as quais, é óbvio, são específicas de uma máquina, de um processador (UCP).

```
(1)Soma          Proc C Value: Palavra
                 (2)Mov  (3) AX,0                (4) Inicializar, zerando



(1) Rótulo
(2) Operação
(3) Operandos (registrador AX e o valor 0)
(4) Comentário
```

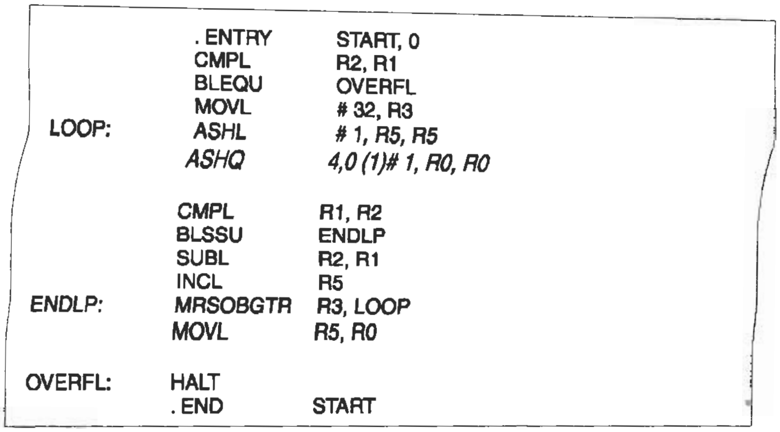
Figura 6.27 Partes componentes de uma linha de código Assembly.

Desta forma, um montador (*assembler*), desenvolvido para entender instruções Assembly do processador Pentium e convertê-las em código binário (instrução de máquina) daquele processador, não poderá ser utilizado para programas escritos na linguagem Assembly do processador Power PC. O nome das instruções Assembly do Power PC é diferente do nome de uma instrução semelhante no Pentium. Também o formato das instruções é diferente e, principalmente, o código gerado para cada processador é diferente, porque seu conjunto de instruções é bastante diferente.

```
CHNGSTR   SAVE    (14,12)
          BALR    12,0
          USING   *,12
          LM      4, 5, 0 (1)     ADDRESS OF STRING IN REG. 4
          L       5,0 (5)         LENGTH IN REG. 5
          LA      6,1
          LA      7,0 (4,5)
          S       7, = f'1'       ADDRESS OF LAST BYTE IN STRING
LOOP      CLI     0 (4), C'#'     TEST FOR #
          BE      REPL1
          CLI     0 (4) C'&&'     TEST FOR &
          BE      REPL2
          CLI     0 (4), C'%'     TEST FOR %
          BE      REPL3
          CLI     0 (4), C'<'     TEST FOR <
          BNE     LOOPCONT
          MVI     0 (4), C')'     IF <, REPLACE BY )
          B       LOOPCONT
REPL1     MVI     0 (4), C' ='    IF # REPLACE BY =
          B       LOOPCONT
REPL2     MVI     0 (4), C'+'     IF &, REPLACE BY +
          B       LOOPCONT
REPL3     MVI     0 (4), C'('     IF %, REPLACE BY (
LOOPCONT  BXLE    4,6, LOOP       LOOP CONTROL
          RETURN  (4,12)
```

Figura 6.28 Exemplo de um programa em linguagem de montagem (Assembly) do processador Intel 80486.

A Fig. 6.28 mostra um exemplo de programa escrito na linguagem de montagem dos microprocessadores Intel 80386/80486, enquanto a Fig. 6.29 apresenta um trecho de programa na linguagem de montagem dos antigos sistemas VAX-11.



```
              . ENTRY      START, 0
              CMPL         R2, R1
              BLEQU        OVERFL
              MOVL         # 32, R3
  LOOP:       ASHL         # 1, R5, R5
             ASHQ         4,0 (1)# 1, R0, R0

             CMPL         R1, R2
             BLSSU        ENDLP
             SUBL         R2, R1
             INCL         R5
 ENDLP:      MRSOBGTR     R3, LOOP
             MOVL         R5, R0

 OVERFL:     HALT
             . END        START
```

Figura 6.29 ***Exemplo de um programa em linguagem de montagem (Assembly) do processador VAX-11.***

Embora no passado a linguagem de montagem tenha sido utilizada em grande escala, especialmente no desenvolvimento de programas básicos (sistemas operacionais) e de controle, atualmente isto já não ocorre. Como a linguagem de montagem tem uma relação de 1:1 com a linguagem de máquina, os programas desenvolvidos em linguagem de montagem tendem a ser sempre maiores (têm mais instruções) do que seus equivalentes em linguagem de alto nível.

Além de maiores e, portanto, de consumirem mais tempo no seu desenvolvimento, os programas escritos em linguagem de montagem são consideravelmente mais complexos no seu entendimento do que o seu correspondente programa em linguagem de mais alto nível. Em geral, aqueles programas (em Assembly) são "dedicados" ao seu criador, isto é, somente quem os elaborou tem facilidade de entendê-los (é claro que um programador experiente em linguagem de montagem poderá, depois de um certo tempo e esforço, entender e até modificar um complexo programa em Assembly). Deste modo, a manutenção desses programas pode se tornar um problema para o proprietário do respectivo sistema, seja pelo seu custo, seja pela saída de programadores experientes (especialmente do desenvolvedor — "o criador" do programa).

A pretensa vantagem dessa linguagem era calcada na pressuposição de que, por atuarem diretamente com o hardware, utilizavam de modo mais eficiente os recursos computacionais às vezes escassos. Ou seja, cada instrução de linguagem de montagem deve atuar diretamente com a operação desejada e usando apenas e exclusivamente as células de memória necessárias. Ao contrário, por exemplo, de um programa desenvolvido em linguagem COBOL, cuja compilação (ver Cap. 9), isto é, a conversão para linguagem de máquina, pode utilizar mais código binário do que se ele tivesse sido escrito diretamente em código binário (ou em linguagem de montagem).

Atualmente há linguagens poderosas, como a linguagem C, que não só manipulam estruturas primitivas como também permitem que o programador desenvolva programas mais claros, estruturados e menores, como só as linguagens de programação de alto nível possibilitam. Muitos programas básicos são escritos hoje em linguagem C, como o Windows da Microsoft e o processador de textos Word Perfect.

No Cap. 9, Execução de Programas, vamos procurar explicar como funciona o programa Montador (Assembler), descrevendo o processo de conversão de um programa em linguagem de montagem para o programa equivalente em linguagem de máquina.

6.6 UM POUCO MAIS DE DETALHE

Vamos abordar neste item alguns detalhes sobre a Unidade Central de Processamento, não considerados nos itens anteriores, e que são destinados àqueles com interesses mais específicos.

No item 6.6.1 serão apresentados a organização e o funcionamento mais detalhado da UAL, descrevendo aspectos mais importantes sobre a metodologia *pipelining*, de execução de programas pela UCP no item 6.6.2. No item 6.6.3 serão apresentadas as características básicas do barramento utilizado nos sistemas de computação e no item 6.6.4 serão abordados, também, aspectos relativos ao tipo de controle que pode ser implementado em um processador: por hardware e por microprogramação. Finalmente, no item 6.6.5, serão apresentadas outras considerações sobre os processadores, não abordadas nos itens anteriores e algumas arquiteturas de processadores mais atuais e, no item 6.6.6, serão apresentadas algumas observações e detalhes sobre elementos auxiliares na implementação dos sistemas.

6.6.1 Unidade Aritmética e Lógica — UAL

A UAL é o componente da UCP cuja função reside na realização das operações matemáticas requeridas por instruções de máquina.

Genericamente, podemos esquematizar o posicionamento da UAL em relação aos demais componentes da UCP como mostrado na Fig. 6.30.

Em geral, a UAL é basicamente constituída de circuitos dedicados a realizar operações de soma, praticar operações lógicas AND e OR, de um circuito inversor (NOT) ou complemento, de um circuito para efetuar deslocamentos dos bits de um número e realizar operações de multiplicação.

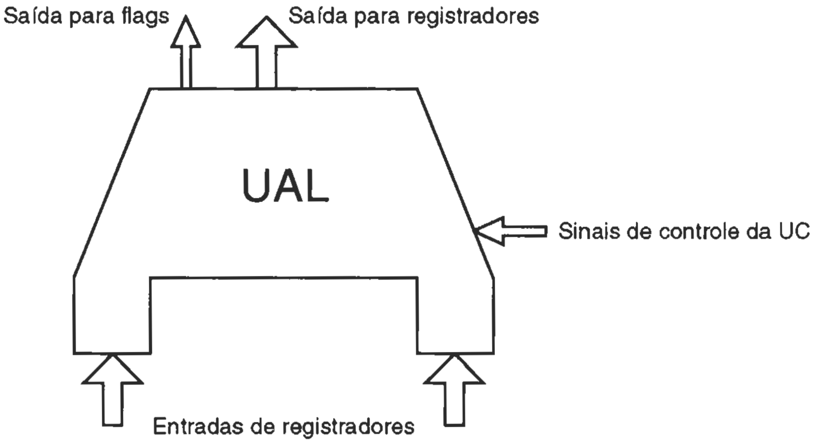


Figura 6.30 Interligação da UAL ao restante da UCP.

As operações matemáticas antes referidas são usualmente as de adição, deslocamento, rotação e operações lógicas, todas essas realizadas sobre dois operandos; e a de complemento, que utiliza apenas um único operando. Nos sistemas mais antigos, havia a possibilidade de utilização de dois tipos de processadores, sendo um deles exclusivamente para realizar operações aritméticas com valores fracionários, representados em ponto flutuante; tais processadores eram denominados co-processadores matemáticos, como o Intel 8087, que funcionava com o Intel 8086/8088 e o Intel 80387, que operava juntamente com o Intel 80386.

A partir do processador Intel 486, a UAL responsável pelas operações em ponto flutuante passou a fazer parte integrante da pastilha do processador. Assim, integradas na mesma pastilha (chip) estavam a UAL responsável pelas operações com valores inteiros e uma UAL para valores fracionários (no caso das arquiteturas dos processadores Pentium há duas UAL escalares, para inteiros, denominadas pela Intel de ALU — Arithmetic and Logic Unit e uma UAL que executa as operações aritméticas de valores fracionários, representados em ponto flutuante, denominadas pela Intel de FPU — Floating Point Unit). Atualmente, todos os processadores se valem deste modelo de arquitetura, integrando na mesma pastilha os dois tipos de UAL.

De modo simples, podemos exemplificar o funcionamento da UAL como um conjunto de circuitos lógicos, utilizados conforme o tipo de operação a ser realizada, que recebem na entrada dois valores (ou um valor apenas se, por exemplo, a operação é de complemento). Esses valores percorrem o circuito lógico determinado pelo sinal da UC e apresentam o resultado na saída, conforme mostrado na Fig. 6.31.

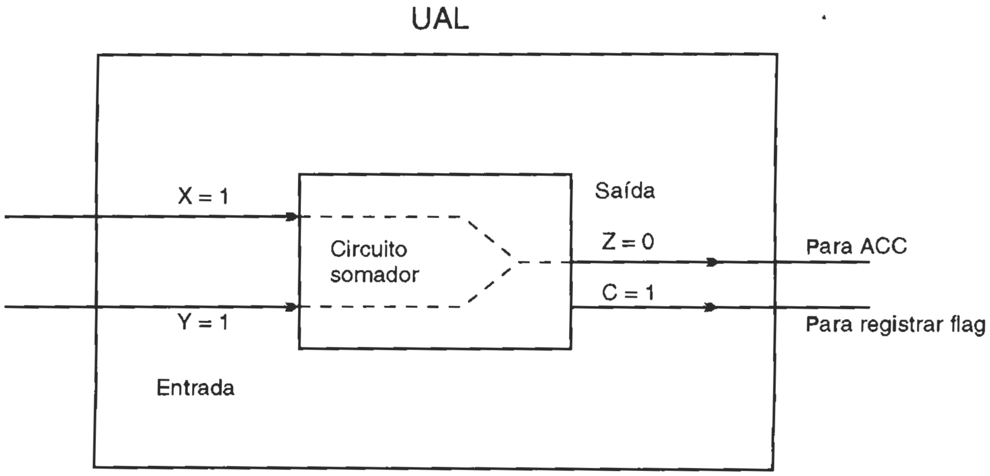
A soma de 2 valores binários, X = 1 e Y = 1, produz como resultado o valor Z = 0 e um bit de "vai 1" ou "carry", cujo valor é igual a 1. Este bit de "vai 1" é, em geral, armazenado em um registrador especial para ser utilizado pelo processador quando necessário (denominado Registrador FLAG em microprocessadores — ver Fig. 6.9).

Pode-se imaginar, ainda, que o tamanho físico do circuito somador tem uma certa influência no tempo em que a operação se realiza, visto que se trata de pura utilização da fórmula: e = vt, onde

e (espaço) é a distância entre a entrada e a saída do circuito;

v é a velocidade do percurso dos sinais elétricos que representam os bits através do circuito que é nominalmente a velocidade da luz, embora na prática seja bem menor que os 300.000 km/s no vácuo; e

t é o tempo de execução da operação, ou seja, o tempo de percurso dos sinais desde a entrada até a saída.



X e Y - valores que serão somados (1 + 1 = 0 e vai 1)
Z - resultado da operação m = 0
C - bit de "vai 1" = 1

Figura 6.31 Funcionamento básico da UAL.

Então, o módulo básico encontrado em uma UAL é o **circuito somador**. Essencialmente, há duas categorias de somadores: o somador parcial ou *half adder* e o somador completo ou *full adder*. Na primeira categoria, a soma é realizada apenas com os dois valores binários envolvidos, sem considerar o "vai 1", que pode ocorrer. No caso do somador completo, o "vai 1" é considerado na soma, que fica, assim, com três parcelas.

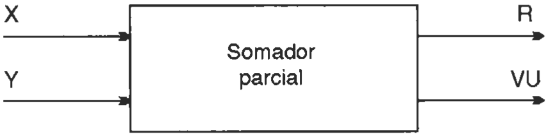
Em seguida, vamos descrever dois exemplos de somadores, de um somador parcial (Fig. 6.32) e de um somador completo (Fig. 6.33).

A função do somador parcial é somar 2 dígitos binários que são recebidos na entrada (X e Y no circuito da Fig. 6.32(a)) e produzir um resultado (o algarismo resultante da soma, R = 0, no exemplo da figura, e um algarismo representativo do "vai 1", VU, também mostrado).

Quando o número a ser somado é composto de vários algarismos, seria inadequado utilizar vários somadores parciais, uma vez que a desvantagem deste tipo de somador é que ele não possui entrada para o "vai 1". Uma soma completa entre 2 valores deve, na realidade, possuir entrada para 3 valores: os 2 algarismos a serem somados e o "vai 1", a ser também somado na mesma operação, mesmo no caso deste valor (do "vai 1") ser igual a 0.

Se somarmos dois números, por exemplo) 1101 e 1100, teremos duas etapas. Na primeira, os números são somados sem considerarmos o "vai 1" que pode ocorrer (na realidade, pode-se considerar, como acontece nos processadores, que o "vai 1" sempre existe, podendo ser igual a 1, quando ele efetivamente acontece, e igual a 0, quando a soma das parcelas não excede o maior algarismo binário).

Operação normal	Operação em partes	
1 1000	1101	
1101	+ 1100	
+ 1100		
11001	00001	soma parcial
	11000	"vai 1"
	11001	soma completa

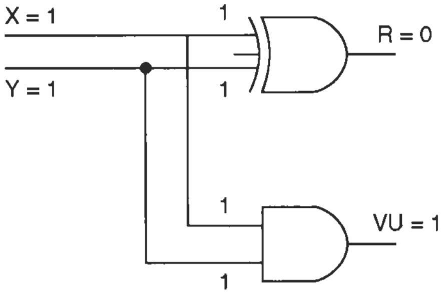


(a) Diagrama em bloco

Entrada		Saída	
X	Y	R	VU
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Tabela verdade

R - resultado
VU - "vai 1"



X + Y = R = 0 , com VU ="vai 1"= 1

1 XOR 1 = 0 (R)
1 and 1 = 1 (VU) } (ver Cap.4)

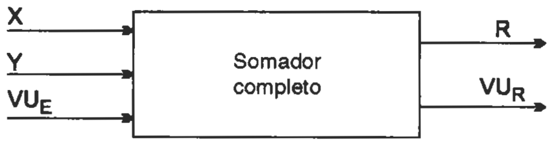
(c) Circuito lógico que implementa um somador parcial. Exemplo da soma da Fig.6.31.

Figura 6.32 Esquema de funcionamento de um somador parcial.

O circuito necessário para realizar a soma do conteúdo de dois registradores deve ter recursos para somar não só os algarismos dos mesmos como também os "vai 1".

A solução encontrada foi o desenvolvimento do somador completo, mostrado na Fig. 6.33.

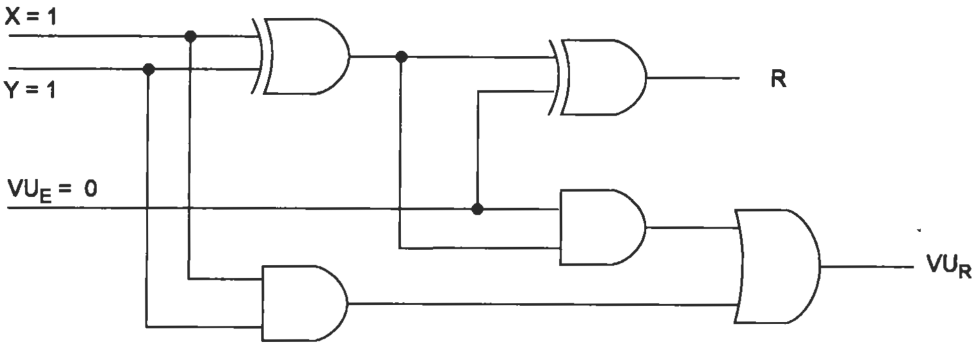
Na Fig. 6.33(a) é apresentado o diagrama em bloco do dispositivo somador, consistindo em 3 entradas: X, Y e VU_E, e 2 saídas: R e VU_R. A Fig. 6.33(b) apresenta a tabela verdade para as 3 entradas e 2 saídas, consistindo nas 8 combinações possíveis. As Figs. 6.33(c) e 6.33(d) mostram o circuito lógico completo com exemplo de sua utilização em dois exemplos de operações de soma. No primeiro caso (Fig. 6.33(c)), adiciona-se X = 1, Y = 1 e $VU_E = 0$ (não há "vai 1"), resultando em: R = 0 e $VU_R = 1$. No outro exemplo, Fig. 6.29(d), adiciona-se o "vai 1" gerado na operação do algarismo anterior e, assim, X = 1, Y = 1 e $VU_E = 1$, resultando em R = 1 e $VU_R = 1$.



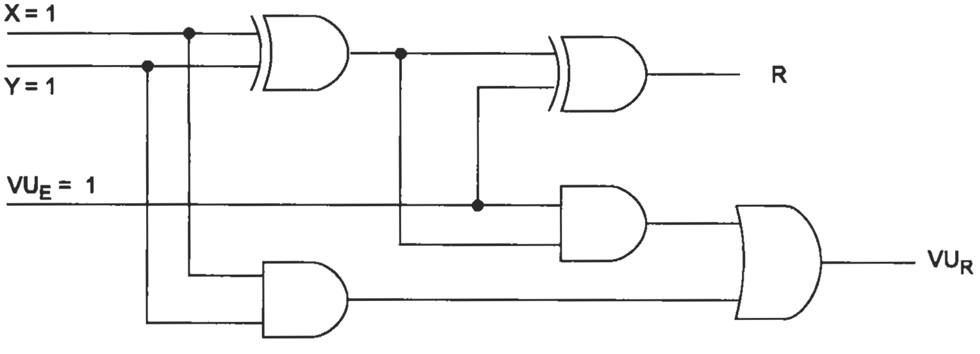
(a) Diagrama em bloco

Entrada			Saída	
X	Y	VU_E	R	VU_R
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1
			0	1

(b) Tabela verdade



(c) Circuito lógico de um somador completo, com exemplo da soma de X = 1, Y = 1, $VU_E = 0$ e resultado R = 0 e $VU_R = 1$



(d) Circuito lógico do somador completo com exemplo da soma de X = 1, Y = 1 e $VU_E = 1$ e resultado: R = 1 e $VU_R = 1$

Figura 6.33 Um somador completo.

O somador completo exemplificado na Fig. 6.33 realiza a soma de dois números que possuem apenas um algarismo, o que, na prática, não tem utilidade, visto que a palavra de dados de um computador tem sempre uma quantidade **n** de bits, sendo que, em geral, **n** >> 1 (a palavra do Pentium PRO é de 32 bits, como também a palavra do processador AMD K-6). Dessa forma, uma UAL é fabricada com um circuito somador capaz de realizar somas de 2 números, cada um com tantos bits quanto o valor da palavra. Na Fig. 6.34 apresentamos um exemplo de um somador paralelo com 4 bits. O propósito deste somador é a adição de dois números binários inteiros com 4 bits cada um.

Os primeiros operandos de entrada foram nomeados como X_0, X_1, X_2 e X_3, e os outros operandos de entrada (2.º operando da soma) como Y_0, Y_1, Y_2 e Y_3.

Consideremos a seguinte adição:

	3 2 1 0
X	0 0 1 1
Y	1 0 1 1
Soma	1 1 1 0

Sendo:
$X_0 = 1$, $X_1 = 1$, $X_2 = 0$ e $X_3 = 0$
$Y_0 = 1$, $Y_1 = 1$, $Y_2 = 0$ e $Y_3 = 1$

Vamos acompanhar a soma de cada par de algarismos (Xi, Yi), utilizando a Fig. 6.34.

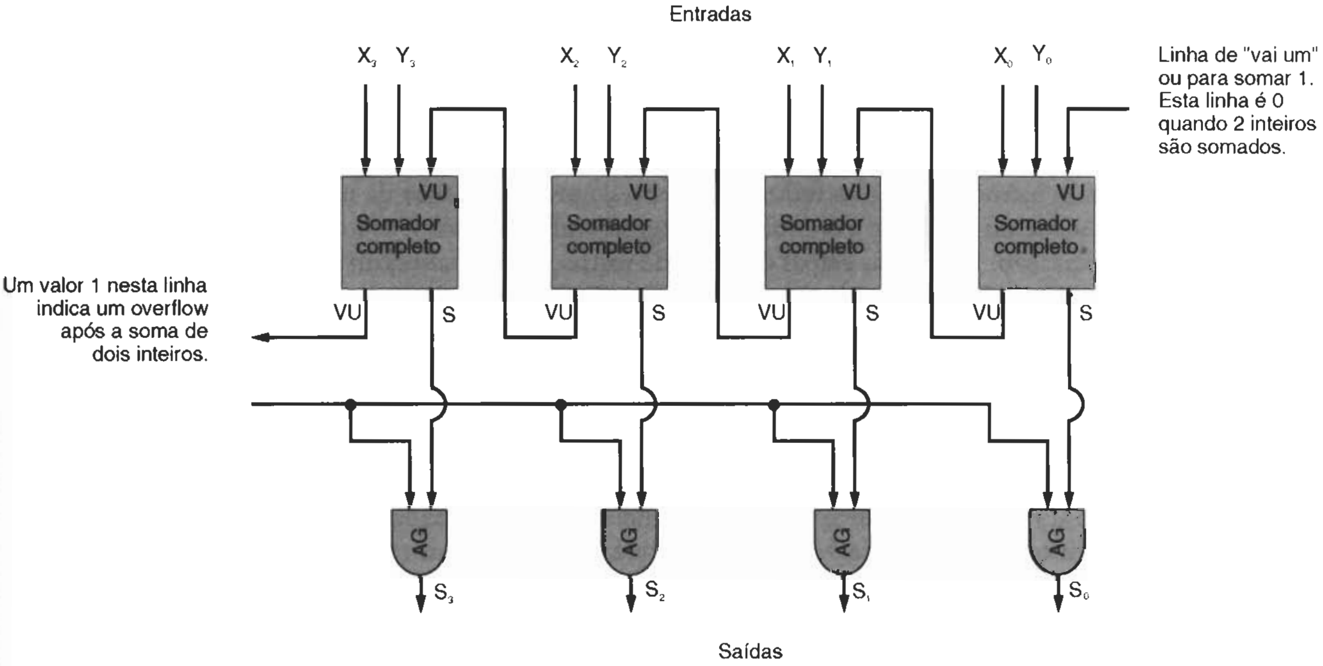

Entradas
X3 Y3
X2 Y2
X1 Y1
X0 Y0
Linha de "vai um" ou para somar 1. Esta linha é 0 quando 2 inteiros são somados.
VU
Somador completo
VU
S
Um valor 1 nesta linha indica um overflow após a soma de dois inteiros.
AG
S3
S2
S1
S0
Saídas

Figura 6.34 Um somador em paralelo, para 4 bits.

A soma dos primeiros algarismos, X_0 e Y_0, não inclui a entrada VU, relativa ao "vai 1" porque não há algarismo anterior, então não pode ainda haver "vai 1". Esta soma, $X_0 = 1$ e $Y_0 = 1$, resulta em um valor 0 e a geração do "vai 1". Este "vai 1" (VU_{S0} é transferido para o 2.° estágio e serve de entrada para VU_{E1} juntamente com $X_1 = 1$ e $Y_1 = 1$). O resultado da soma dos 3 algarismos todos de valor 1 (Fig. 6.32(b)) é igual a 1 ($S_1 = 1$), com geração de "vai 1" ($VU_{S1} = 1$), o qual é encaminhado para a entrada (VU_{E2}) do 3.° estágio.

A soma de $X_2 = 0$, $Y_2 = 0$ e $VU = 1$ resulta em um valor $S_2 = 1$ e $VU = 0$, pois não há "vai 1" (que será $VU = 0$), o qual será enviado para a entrada do último estágio do somador (4.° algarismo), tornando-se $VU = 0$. A soma final será $X_3 = 0$, $Y_3 = 1$ e $VU = 0$, cujo resultado é $S_3 = 1$, sem a existência de "vai 1". Em conseqüência, a linha $VU = 0$, o que indica que o resultado da operação está correto e que não houve estouro de algarismos (*overflow*) (ver item 7.5.4 para uma descrição mais detalhada do conceito de *overflow*).

Se a operação aritmética de adição fosse realizada com os seguintes valores:

```
                    100110
                   +110011
estouro (overflow) → 1011000
```

ocorreria um "vai 1" para um 5.° algarismo; a linha de saída VU seria igual a 1, indicando esta ocorrência, que é um erro na operação devido à limitação de 4 bits do sistema exemplificado (ver item 7.5.4).

O somador descrito não considera o sinal do número, ele apenas soma os algarismos da magnitude. Para considerar o sinal em uma operação dessas, seria necessário um circuito adicional, cuja forma de construção depende do modo como os números negativos são representados na UAL: podem ser representados em sinal e magnitude, ou ainda em complemento a 1 ou complemento a 2 (ver item 7.2.3 para uma completa descrição destes métodos).

Embora o exemplo apresentado se referisse a um somador com 4 bits, ele pode perfeitamente servir para um somador de 8, 10, 16 ou qualquer quantidade de bits.

6.6.2 Metodologia Tipo Linha de Montagem ou *Pipelining*

Ao descrever o funcionamento da UCP na realização de seus ciclos de instrução foi observado que, embora o ciclo de instrução seja composto de várias etapas (ver o fluxo da Fig. 6.20), ele é realizado basicamente de forma seqüencial, isto é, uma etapa se inicia após a conclusão da anterior. Este procedimento segue tipicamente o conceito de arquitetura de um sistema de computação proposto por von Neumann (ver Cap. 1) e que já foi definido como sendo do tipo SISD.

UCPs deste tipo vêm sendo usadas desde as primeiras gerações de computadores, e muitos aperfeiçoamentos tecnológicos foram introduzidos para reduzir o tempo de processamento de uma instrução, entre os quais o aumento da velocidade do relógio e a tecnologia de semicondutor, com seus sucessivos melhoramentos em fabricação e miniaturização. Porém, as etapas do ciclo de instrução permaneciam sendo realizadas seqüencialmente, como mostrado na Fig. 6.35. Na figura podemos observar este fato e verificar que o tempo total de execução da instrução é a soma dos tempos gastos em cada etapa.

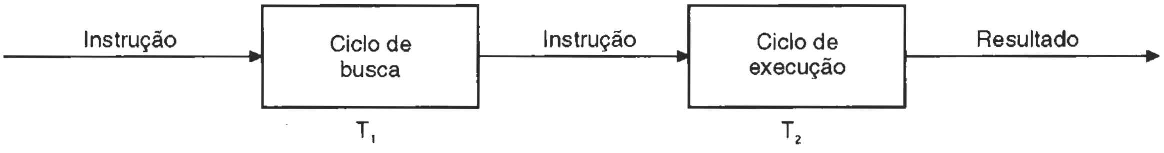
O processo é semelhante ao da fabricação de um automóvel, caso ainda existisse uma fábrica que funcionasse em montagem seqüencial: primeiro, a carroceria (gastaria um tempo T_1), depois o motor (tempo T_2), em seguida as rodas (tempo T_3), depois a parte elétrica (T_4), os bancos e espelhos (T_5) e o acabamento final (T_6). Cada carro ficaria pronto em um tempo $T = T_1+T_2+T_3+T_4+T_5+T_6$ e somente com a total conclusão de um carro (tempo T) é que um outro automóvel iniciaria sua montagem. A fábrica produziria, em um turno de X horas, um total máximo de X/T carros, o que sempre seria um valor pequeno.

Uma outra metodologia, inventada no início do século XX por Henry Ford e atualmente seguida pela maioria das indústrias, consiste em dividir o processo de fabricação em estágios independentes (que, por isso, podem se superpor uns aos outros, no tempo), denominando-se linha de montagem (em inglês se usa o termo "pipeline", pois um item pode iniciar sua fabricação antes que um item anterior termine sua montagem). Em computação, a metodologia de construção da UCP composta de estágios permitiu que, também nestes sistemas, se adotasse esta técnica.

Vamos descrever um pouco mais a metodologia. A característica fundamental do processo de *pipelining* (vamos usar este termo em vez de linha de montagem, dada a sua popularidade entre os profissionais de Informática) reside em duas premissas básicas:

a) a divisão do processo (seja o de fabricação de um automóvel, de uma TV ou o ciclo de uma instrução na UCP) em estágios de realização independentes um do outro; e

b) um novo produto (automóvel, TV ou instrução) inicia seu processo de fabricação ou execução antes do anterior concluir seu processo.



T = tempo de execução da instrução = $T_1 + T_2$

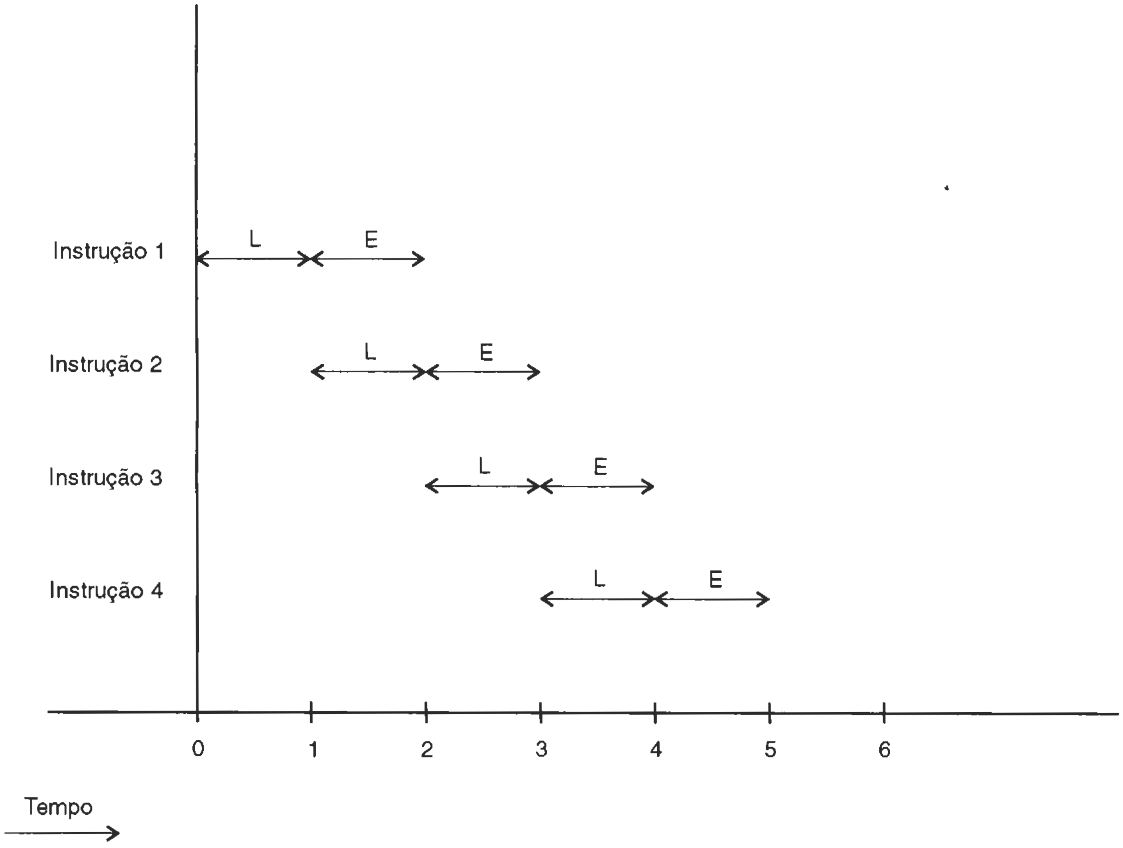
Ciclo de busca = leitura da instrução, incremento do CI

Ciclo de execução = decodificação, busca do operando, execução da operação

Figura 6.35 Ciclo de instrução em execução seqüencial.

Suponhamos que o processo de realização do ciclo de uma instrução seja dividido em dois estágios: o da *leitura ou busca da instrução* e o *da execução da instrução* lida.

Neste caso, para ler uma instrução, é necessário um acesso à memória, mas para executar a instrução nem sempre é necessário acessar a memória (por exemplo, na decodificação e na execução da operação não há acessos à memória). Portanto, é possível ler uma instrução, utilizando-se dos circuitos de um estágio, e transferir esta instrução para o estágio de execução. E, durante o período em que, neste estágio, não há atividade com a memória, pode-se ativar o estágio de leitura para buscar uma nova instrução e continuar o processo com novas instruções, como mostrado na Fig. 6.36. Esta atividade de busca de nova instrução antes da conclusão da anterior é conhecida como pré-busca (*pre-fetch*).



L - leitura (busca) da instrução

E - execução

hipótese : tempo de L = tempo de E

Figura 6.36 Exemplo de execução de uma instrução com metodologia *pipelining* com 2 estágios.

Considerando que cada estágio gaste o mesmo período de tempo T (L gasta T unidades de tempo e E gasta T unidades de tempo), então uma instrução executada de forma seqüencial gastaria 2T unidades de tempo para ser concluída, ao passo que, com o processo *pipelining*, uma instrução é conhecida a cada tempo T, ou seja, o tempo de execução se reduziria pela metade.

Na realidade, pode não haver muita produtividade em um sistema destes (*pipelining* com 2 estágios), porque:

a) o *tempo de realização do estágio L não é igual ao do estágio E.* Em geral, a execução consome mais tempo, devido principalmente à etapa de busca de operando. E, portanto, na maioria do tempo de execução (E) pode não ser possível haver outra busca de instrução;

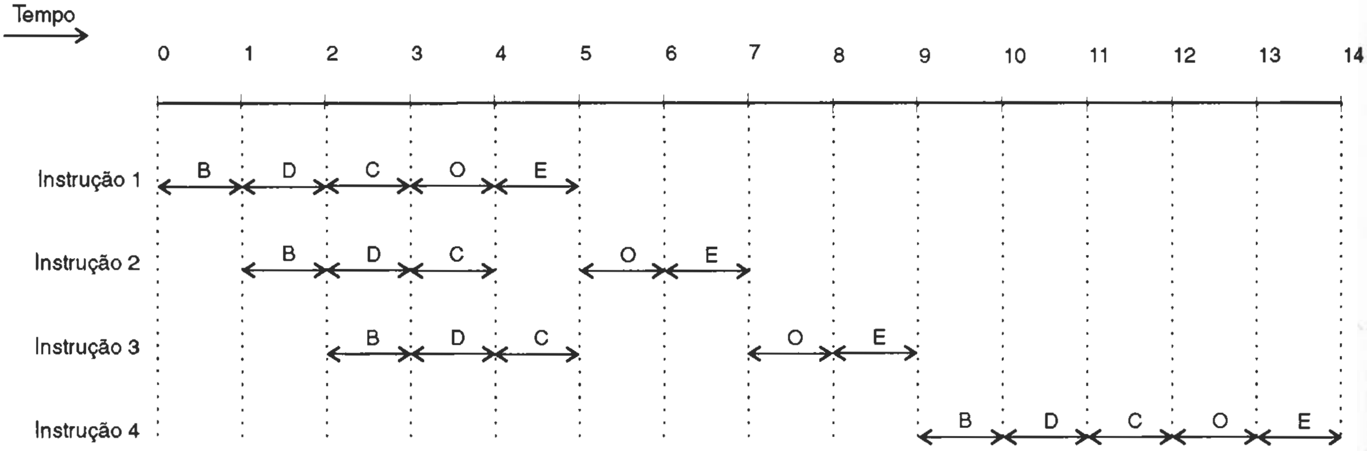
b) *pode não ser possível buscar nova instrução antes da execução completa da anterior.* Em uma instrução de desvio, por exemplo, o endereço de desvio só é conhecido após a execução da operação e, nesse caso, não há como "buscar" uma nova instrução durante o estágio de execução. Assim, o estágio de busca não será, neste caso, superposto ao de execução, e o de execução da instrução seguinte também vai acontecer somente após sua busca. Ou seja, nada se ganhou em termos de tempo.

Apesar disso, os processadores Intel 8086/8088 foram projetados com 2 estágios de processamento, cada um realizado por uma unidade independente: a unidade de interface do barramento (BIU — Bus Interface Unit) e a unidade de execução (EU — Execution Unit), conforme mostrado mais adiante ainda neste capítulo. O sistema funcionava muito bem para o seu padrão.

No entanto, em face dos problemas antes mencionados para o caso de um pipeline de 2 estágios (tempos desiguais de duração dos estágios e caso das instruções de desvio), com o propósito de obter-se maior produtividade e rapidez do sistema, deve-se construir a UCP com mais estágios. Quanto maior a quantidade de estágios, mais superposição e aumento de velocidade. É importante ressaltar que o tempo de duração de cada estágio deve ser o mais semelhante possível, de modo que um estágio não tenha que esperar o término do outro para iniciar a execução seguinte.

A Fig. 6.37 apresenta um esquema de execução de instruções utilizando o método *pipelining* com 5 estágios. O ciclo de cada instrução é executado nos estágios de Busca (B), de Decodificação (D), de Cálculo do Endereço dos Operandos (C), de Busca dos Operandos (D) e de Execução da Operação (E).

Considerando a hipótese definida, de estágios com tempo de duração igual (quanto mais estágios, maior a probabilidade de se obter tempos iguais em cada estágio), podemos verificar, pelo diagrama de tempo da Fig. 6.37, que o tempo total de execução das 4 instruções do exemplo é de 14 unidades de tempo. Se as 4 instru-



B - busca de instrução (com acesso à memória)

D - decodificação do código de operação (sem acesso à memória)

C - célula do endereço dos operandos (sem acesso à memória)

O - busca do(s) operando(s) (com acesso à memória)

E - execução da operação (com acesso à memória - nem sempre é verdadeira, mas neste exemplo sim)

Figura 6.37 Exemplo de ciclo de instrução do tipo *pipelining* com 5 estágios.

ções fossem executadas em um processador seqüencial, consumiriam 20 unidades de tempo (4 instruções × 5 unidades de tempo cada = 20). No diagrama da Fig. 6.37 consideramos que:

- somente um acesso à memória pode ser realizado de cada vez, isto é, durante a execução do estágio de busca, B, não pode ser realizado, por exemplo, o estágio 0 — busca de operandos;
- no estágio de execução da operação (E) há acesso à memória para armazenar o resultado da operação, embora saibamos que nem sempre isto é verdadeiro;
- todos os estágios são realizados em um período de tempo igual. Se isto não acontecer na prática, poderá haver alguma espera entre o término de um estágio e o início de outro. Por exemplo, se o estágio de busca de operando (D) dura um pouco mais que os outros estágios, então o de execução se inicia depois, e assim sucessivamente. No exemplo da Fig. 6.37, isto possivelmente acarretaria o acréscimo de, pelo menos, mais uma unidade de tempo no total.

Um dos problemas mais críticos quando se projeta um esquema de *pipeline* reside na necessidade de se evitar atrasos no processamento dos estágios, de modo a se manter um fluxo contínuo das instruções no seu percurso de um estágio para outro. Um dos fatores que mais complicam a obtenção deste propósito é o tratamento dado às instruções de desvio, especialmente às de desvio condicional.

A instrução de desvio condicional, mostrada no exemplo da Fig. 6.19, JP Op. indica que

se ACC > 0, então: CI ← Op.

Instruções de desvio em processadores reais (Jump ou Branch) seguem essencialmente o mesmo modelo, ou seja: o processador busca a instrução, decodifica seu código de operação e vai testar o valor do ACC (pelo valor do específico bit do registrador de Flags). Se o valor armazenado no ACC for positivo, a próxima instrução a ser executada estará no endereço especificado na instrução. Porém, se o valor armazenado não for positivo, o endereço da próxima instrução será diferente.

Então, em um desvio condicional, não há meios de se conhecer o endereço da próxima instrução, a não ser após o teste da condição definida na instrução; se o teste for verdade, o endereço é um (de desvio); caso contrário, não haverá desvio e o endereço é o seguinte (calculado pelo incremento natural do CI). E o teste de validade da condição somente é realizado durante o estágio de execução da operação.

Se, no exemplo da Fig. 6.37, a instrução 2 for uma instrução de desvio condicional, o estágio B (busca da instrução) da instrução 3 não poderá ser iniciado na unidade de tempo 2, como mostrado na figura, porque naquele instante o endereço da próxima instrução ainda não é conhecido, o que somente poderá acontecer após a execução da operação (estágio E). É durante a fase de execução que se realiza o teste de condição e, dependendo do resultado (Falso ou Verdadeiro), o endereço de desvio (endereço da próxima instrução) será um (a instrução imediatamente seguinte) ou outro (instrução fora da seqüência). Neste caso, haverá um atraso no fluxo de processamento das instruções.

Ao longo do tempo, várias alternativas vêm sendo analisadas, conduzindo a algumas formas diferentes de implementação de processamento *pipelining*. Entre essas alternativas podemos citar:

- *busca de ambas as instruções antecipadamente* (*pre-fetch*) — por esse método, o sistema, ao reconhecer uma instrução de desvio condicional (estágio de decodificação do código de operação), realiza a leitura da instrução seguinte na seqüência e a leitura da instrução se o desvio ocorrer. De modo que, ao ser obtido o resultado do teste de condição, ambas as instruções já estarão disponíveis e o fluxo de processamento não será atrasado.
- *previsão da ocorrência ou não do desvio* — consiste na determinação da ocorrência ou não do desvio. Tendo em vista o tipo das instruções anteriores, o sistema estima se o desvio ocorrerá ou não. Isto pode ser realizado com o auxílio do programa compilado e de lógica especial na UCP (ver item 9.3 para conceituação e funcionamento de compiladores).

Por exemplo, numa seqüência de instruções de um *loop*, ocorrerá desvio para sua reinicialização em 99% das vezes, exceto no final do *loop*.

O processador Intel Pentium emprega este último método; um dispositivo no processador mantém uma história dos desvios anteriores e decide se haverá ou não o desvio na instrução corrente baseado nos dados passados.

Processadores com tecnologia RISC (ver Cap. 11) empregam *pipelining* em larga escala, de modo a concluir a execução de uma instrução a cada ciclo de relógio (e até duas ou mais instruções por ciclo, como os processadores Power PC, da IBM /Motorola /Apple e Alpha, da DEC).

6.6.3 Barramento

Neste item, vamos procurar expandir um pouco mais o conhecimento sobre o barramento utilizado nos computadores, descrevendo os tipos mais empregados e seu funcionamento básico.

Conforme já mencionamos (é importante repetir, consolidando os conceitos e definições expostos no Cap. 2), o *barramento* de um sistema de computação é o elemento responsável pela interligação dos demais componentes, conduzindo de modo sincronizado o fluxo de informações de uns para os outros — dados, endereços e sinais de controle — de acordo com uma programação de atividades previamente definida na UC.

Conforme já mencionado anteriormente, o barramento de dados consiste em múltiplas linhas condutoras, cada uma permitindo a passagem de um bit de informação (seja de uma instrução ou de um dado). Tais barramentos possuem diferentes tamanhos (quantidade de bits), dependendo do modelo do processador utilizado. Valores típicos são 8, 16, 32, 64 e 128 bits.

O barramento de endereços é utilizado para o processador indicar de onde quer ler (buscar) um dado ou para onde deseja gravá-lo. Normalmente, a quantidade de bits de um endereço especifica a máxima capacidade de um módulo de memória principal (ver Cap. 5). Mas o valor binário colocado no barramento de endereços também pode representar o endereço de um dispositivo de entrada ou de saída. Em geral, um ou mais bits do endereço, mostram se o endereço indicado se refere a um módulo de memória principal ou de um dispositivo de E/S.

O barramento de controle é constituído de inúmeras linhas pelas quais fluem sinais específicos da programação do sistema. Sinais comumente empregados nos barramentos de controle são:

- leitura de dados (memory read) — sinaliza para o controlador de memória decodificar o endereço colocado no barramento de endereços e transferir o conteúdo da(s) célula(s) para o barramento de dados.
- escrita de dados (memory write) — sinaliza para o controlador de memória decodificar o endereço colocado no barramento de endereços e transferir o conteúdo do barramento de dados para a(s) célula(s) especificada(s).
- leitura de E/S (I/O read) — processo semelhante ao de leitura de dados da memória.
- escrita de E/S (I/O write) — processo semelhante ao de escrita de dados na memória.
- certificação de transferência de dados (transfer ACK) — o dispositivo acusa o término da transferência para a UCP.
- pedido de interrupção (interrupt request) — indica ocorrência de uma interrupção.
- relógio (clock) — por onde passam os pulsos de sincronização dos eventos durante o funcionamento do sistema.

A Fig. 6.38 mostra o esquema lógico de um tipo de barramento (múltiplos barramentos, um para cada tipo de informação que flui durante o processo de execução de uma instrução — bits dos endereços, bits dos dados e sinais de controle), enquanto a Fig. 6.39 mostra o exemplo de um cabo de ligação (apresentação física de um barramento) entre um periférico e um processador, constituído de diversos fios paralelos bem próximos uns dos outros, cada um conduzindo um bit da informação que está sendo transferida (ou um sinal de controle diferente).

Um dos aspectos fundamentais de um barramento é sua capacidade de compartilhamento pelos diversos componentes interconectados. Assim, por exemplo, em um barramento como o mostrado na Fig. 6.38, que interliga UCP, memória e periféricos, vemos que todos esses elementos compartilham o mesmo caminho e, por essa razão, somente um conjunto de bits pode passar de cada vez. A programação e sincronização desse processo é crucial para o correto funcionamento do sistema.

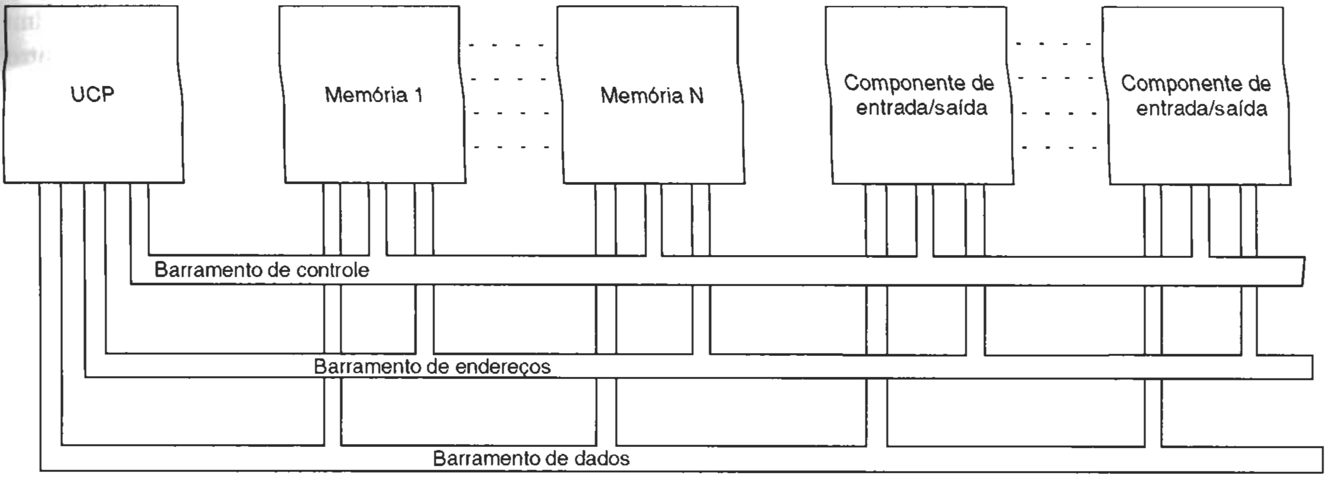


Figura 6.38 Exemplo de esquema de barramento externo de um sistema de computação, constituído de elementos individuais para cada tipo de informação (dados, endereços e controle).

Uma observação deve ser feita neste ponto: o modelo mostrado na Fig. 6.38, de um único barramento de dados, endereços e controle, interconectando todos os componentes do computador, não é mais empregado. As diferentes características entre os diversos componentes, principalmente periféricos (a velocidade de uma transferência de dados de um teclado é muitas vezes menor que a velocidade de transferência de dados de um disco magnético) levou os projetistas de sistemas de computação a criarem diversos tipos de barramento, cada um com taxas de transferência de bits diferentes e apropriadas às velocidades dos componentes interconectados, sendo os barramentos organizados de forma hierárquica.

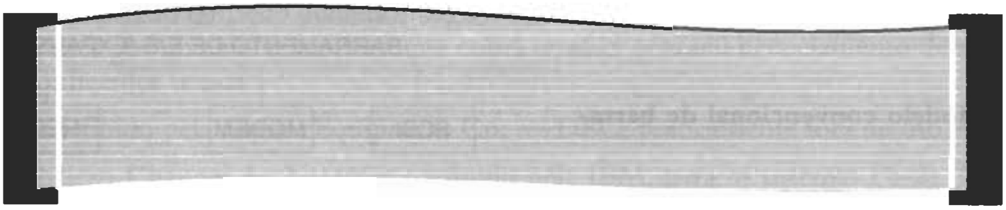


Figura 6.39 Exemplo de um barramento (um cabo paralelo de ligação entre UCP e periférico).

Atualmente os modelos de organização de sistemas de computação adotados pelos fabricantes possuem diferentes tipos de barramento (ver Fig. 6.40):

1. **Barramento local** — é o barramento de maior velocidade de transferência de dados, funcionando normalmente na mesma freqüência do relógio do processador. Este barramento costuma interligar o processador aos dispositivos de maior velocidade (para não atrasar as operações do processador), que são a memória cache e a memória principal.
2. **Barramento do sistema** — alguns fabricantes adotam o modelo em que o barramento local interliga o processador à memória cache e esta se interliga aos módulos de memória principal (RAM) por um outro barramento denominado barramento do sistema, de modo a não permitir acesso do processador diretamente à memória principal. Uma interface de controle sincroniza o acesso entre as memórias.
3. **Barramento de expansão** — onde se interligam os diversos dispositivos de E/S, como discos magnéticos, vídeos, impressoras, DVDs, CD-ROMs, etc. Este barramento se conecta ao barramento do sistema por interfaces de controle (costumam ser conhecidas como pontes ou bridges), que sincronizam as diferentes velocidades dos barramentos. Devido às diferentes e acentuadas velocidades de funcionamento dos dispositivos atuais de E/S, os fabricantes de sistemas de computação têm criado alternativas para

aumentar o desempenho nas transferências de dados, separando o barramento de expansão em dois, um de mais alta velocidade, para dispositivos de E/S rápidos (máquinas SCSI, redes, placas gráficas), e outro de menor velocidade para os modems, dispositivos seriais, como o teclado e mouse. A Fig. 6.40(b) mostra um exemplo desta tecnologia.

A *largura (ou tamanho)* de um barramento é uma unidade de medida que caracteriza a quantidade de informações (bits em geral) que pode fluir simultaneamente pelo barramento. No caso de fiação, consiste na quantidade de fios paralelos existentes no barramento, ao passo que, em circuitos impressos (placas), consiste nos traços impressos na placa com material condutor, por onde flui a corrente elétrica (o cabo mostrado na Fig. 6.35 possui um *tamanho* correspondente aos vários fios que o compõem).

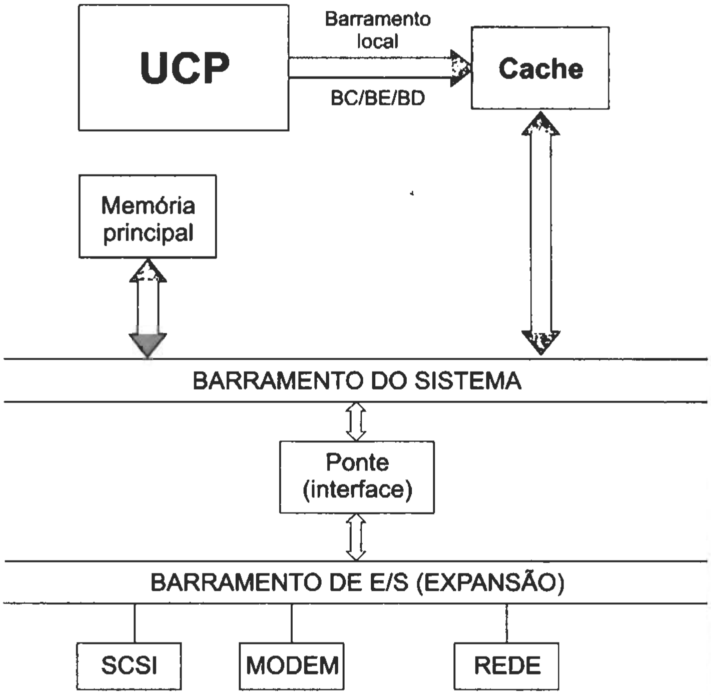

UCP
Barramento local
BC/BE/BD
Cache
Memória principal
BARRAMENTO DO SISTEMA
Ponte (interface)
BARRAMENTO DE E/S (EXPANSÃO)
SCSI
MODEM
REDE

Figura 6.40(a) Modelo convencional de barramento.

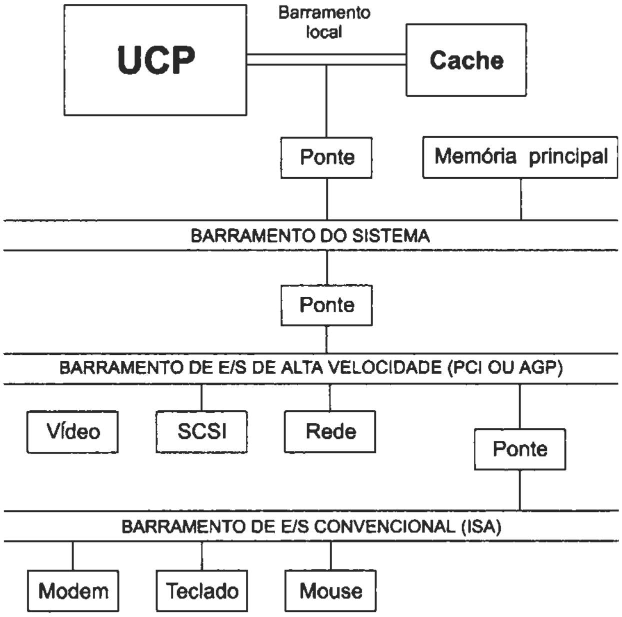

UCP
Barramento local
Cache
Ponte
Memória principal
BARRAMENTO DO SISTEMA
Ponte
BARRAMENTO DE E/S DE ALTA VELOCIDADE (PCI OU AGP)
Vídeo
SCSI
Rede
Ponte
BARRAMENTO DE E/S CONVENCIONAL (ISA)
Modem
Teclado
Mouse

Figura 6.40(b) Modelo aperfeiçoado de barramento (mais recente e de maior desempenho).

Esta largura (ou tamanho), maior ou menor, se constitui também em um dos elementos que afetam a medida de desempenho de um sistema, juntamente com a duração de cada bit ou sinal. A taxa de transferência, que é, em geral, especificada em bits (ou K bits, M bits, etc.) por segundo, depende fundamentalmente da largura do barramento.

O intervalo de tempo requerido para mover um grupo de bits (tantos quanto a quantidade de bits definida pela largura do barramento) ao longo do barramento é denominado *ciclo de tempo* do barramento ou simplesmente *ciclo do barramento* (*bus cycle*), modo análogo ao que definimos para o *ciclo do processador* e para o *ciclo de memória.*

Para entender o funcionamento do barramento de um computador, é preciso enfatizar o aspecto de compartilhamento que caracteriza aquele componente, ou seja, como um barramento interliga diversos componentes (seja barramento interno ou externo); as informações só podem fluir uma de cada vez, senão haverá colisão entre os sinais elétricos e o resultado será ininteligível, qualquer que seja o destinatário.

Em outras palavras, se a memória principal está enviando dados para a memória secundária (componente de E/S, como um disco magnético, por exemplo), os demais componentes têm que esperar a liberação do barramento para utilizá-lo.

Este compartilhamento (um caminho para vários usuários) implica a necessidade de definição de regras bem explícitas de acesso ao barramento por um usuário (quando acessar, como acessar, como terminar) e de comunicação entre eles (como interrogar um componente destinatário, que resposta deve ser enviada, quanto dura a comunicação, etc.). Estas regras costumam ser denominadas protocolos, sendo, no caso, protocolos do barramento, os quais são usualmente implementados através de sinais de controle e exata sincronização entre eles. Assim, o barramento não se constitui tão-somente na fiação já mencionada, mas também na unidade de controle do barramento, que administra o acesso e as transferências (implementação do protocolo adotado).

Para evitar que cada fabricante de UCP crie seu próprio protocolo de barramento com características diferentes dos demais (e, com isso, componentes fabricados por terceiros tenham dificuldade de se conectar à UCP), os próprios fabricantes têm procurado criar uma padronização na definição de protocolos (embora o sucesso total ainda esteja longe — um só padrão em todo o mercado). Ao longo do tempo vários protocolos de barramento de expansão têm sido definidos; alguns tiveram pouca aceitação, outros não, alguns são proprietários (são definidos por uma única empresa que cobra *royalties* pelo seu licenciamento de uso) e outros não. Entre os mais conhecidos temos:

- UNIBUS (definido pela Digital Equipment Co. — DEC, praticamente fora de uso.
- MCA — Micro Channel Architecture (definido pela IBM, para os sistemas PS-2). Nunca conseguiu adoção por outro fabricante, nem mesmo a IBM o adotou por completo, tendo sido abandonado.
- ISA — Industry Standard Adapter (definido pela IBM para o PC-AT e adotado por toda a indústria). Apesar de possuir uma taxa de transferência baixa, tem ainda sido adotado para os barramentos de periféricos de baixa velocidade. Os sistemas atuais costumam empregar algumas portas para periféricos com o modelo ISA.
- EISA — Extended ISA (definido por um grupo de fabricantes para se opor ao MCA da IBM). Não conseguiu ser implantado devido a diversos problemas de especificação, sendo praticamente abandonado pelos fabricantes.
- PCI — Peripheral Component Interconnect — desenvolvido pela Intel, tornando-se quase um padrão para todo o mercado, como barramento de E/S de alta velocidade. Permite transferência de dados em 32 e 64 bits a velocidades de 33 MHz e de 66 MHz, no máximo. Interconecta-se ao barramento local e a outro barramento, tipo ISA, através de um circuito para compatibilizar as diferentes características entre eles. Estes circuitos chamam-se pontes (bridges).
- USB — Universal Serial Bus — tem a particular função de permitir a conexão de muitos periféricos simultaneamente (pode-se conectar até 127 dispositivos em um barramento USB) ao barramento e este, por uma única tomada, se conecta à placa-mãe.
- AGP — Accelerated Graphics Port — barramento desenvolvido por vários fabricantes, porém liderados pela Intel, com o propósito de acelerar as transferências de dados do vídeo para a memória, especialmen-

te dados para 3D. Trata-se, pois, de um barramento específico (para vídeo), não genérico, porém de alta velocidade de transferência por ligar vídeo diretamente à memória principal.

Para se projetar um barramento a ser utilizado em determinado sistema de computação, devem-se considerar alguns detalhes:

a) *Método de controle do acesso ao barramento.* Um dos métodos mais empregados é o de mestre/escravo (*master/slave*). Um dos componentes (em geral é a UCP) é o mestre. É o único que pode acessar o barramento, seja para colocar informações para um determinado componente (escrita), seja para obter informações de um outro componente (leitura). Os demais componentes são os escravos.

Este método tem a vantagem de ser simples e barato de implementar, mas possui uma grande desvantagem: toda a comunicação é realizada via mestre (UCP). Se há volume de dados envolvido nas transferências, pode ocorrer um grande gargalo no funcionamento do barramento, como, por exemplo, durante a transferência de dados entre memória secundária e memória principal.

Uma alternativa para eliminar a desvantagem deste método consiste em se adotarem múltiplos mestres no sistema e não um só para todo o sistema. Esse novo método emprega um dispositivo denominado Acesso Direto à Memória ou DMA — Direct Memory Access (utilizado pela maioria dos sistemas atuais). Ele se caracteriza pela possibilidade de um dos componentes que compartilham o barramento poder ganhar seu controle para transmitir ou receber informações.

Este método, ainda que alivie a UCP para realizar outras tarefas enquanto uma transferência esteja sendo realizada (por exemplo, entre a memória principal e a memória secundária, pelo DMA), requer maior complexidade nos circuitos de controle do barramento e não componentes do sistema para administrar as diversas solicitações simultâneas de acesso ao barramento e os correspondentes sinais de controle.

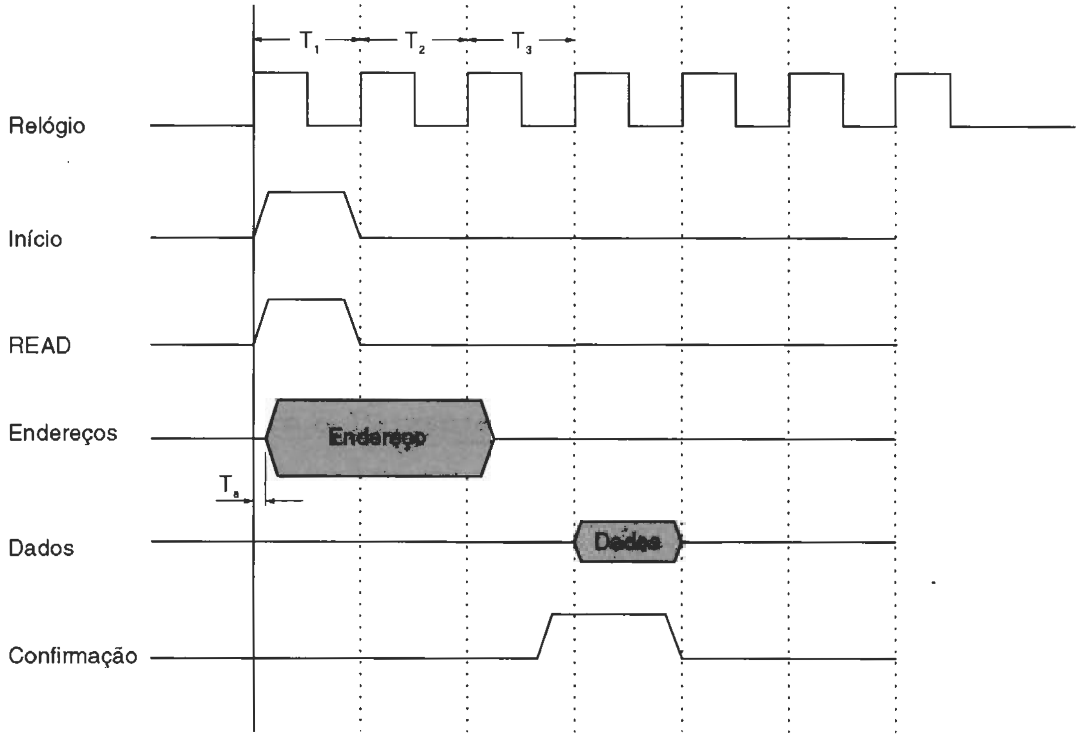
b) *Tipo de sincronização nas operações com o barramento.* Um outro aspecto de projeto de barramento refere-se ao modo pelo qual os eventos são coordenados no barramento. Há duas técnicas disponíveis:

- operação síncrona;
- operação assíncrona.

Com a *operação síncrona*, os pulsos emitidos pelo relógio regulam o aparecimento/desaparecimento dos sinais nas diversas linhas do barramento. Isto é, o relógio sincroniza o funcionamento do barramento e a ocorrência e duração de todos os eventos. Para tanto, o barramento de controle possui uma linha para o relógio, por onde circulam os pulsos gerados por aquele dispositivo, sendo cada pulso denominado ciclo de relógio ou ciclo do barramento (*bus cycle*). Um relógio de 25 MHz tem, por exemplo, um ciclo de barramento de 40 ns.

A Fig. 6.41 mostra um exemplo de uma operação de leitura sendo realizada em um barramento síncrono: as linhas do relógio (com ciclos de barramento iguais a T_1, T_2, T_3, etc.); a linha de início, que é ativada durante um ciclo, para indicar que foram colocados endereço e sinal de controle no barramento (quando isto ocorrer); as linhas de endereço (uma linha dupla para indicar que são múltiplas linhas) e de dados (também múltiplas); e a linha de confirmação (colocada pelo dispositivo acionado para indicar quem cumpriu).

O *ciclo de leitura* começa (como qualquer evento) no início de um pulso de relógio (na transição de 0 para 1 do início da onda quadrada) pela colocação do endereço de leitura no barramento de endereços, emissão de um sinal de leitura (linha READ alta) e alerta desses passos elevando o nível da linha de início. Isto ocorre no primeiro ciclo de barramento, T_1. Durante o período de relógio seguinte, ciclo de barramento T_2, nada acontece no barramento, mas a memória usa este tempo para decodificar o endereço e colocar os dados no barramento, emitindo o sinal de confirmação durante este período. Após o início do 3.° e último ciclo de barramento, T_3, a UCP transfere os bits que estão no barramento de dados para o RDM. Se a memória não conseguir (porque é lenta) decodificar o endereço e colocar os bits de dados no barramento correspondente no tempo requerido, isto é, entre o instante em que o sinal READ e Início foram detectados e o instante (algum tempo depois do início de T_3) em que o dado é colocado efetivamente no barramento, a UCP precisa esperar mais e, para isso, a memória ativa uma linha de espera (WAIT), não mostrada na figura. É o que se chama de *estado de espera* (*wait state*), mencionado no Cap. 5. Haverá tantos ciclos de barramento para espera quantos a memória precisar. Assim que estiver pronta para colocar os dados no barramento, ela desativa a linha de espera.



T_a - tempo de atraso da colocação do endereço na linha.

Figura 6.41 Operação de leitura em um barramento síncrono.

No exemplo mostrado, um sistema com relógio de 25 MHz gastaria três ciclos, no mínimo, para realizar a leitura, cerca de 120 ns, caso não houvesse estados de espera no meio.

Já na operação do *tipo assíncrona* não há relógio sincronizador, nem eventos com duração certa de um ciclo de barramento (já que não há relógio). É claro que quando nos referimos à ausência de relógio é apenas com relação a não haver pulso de relógio para o barramento.

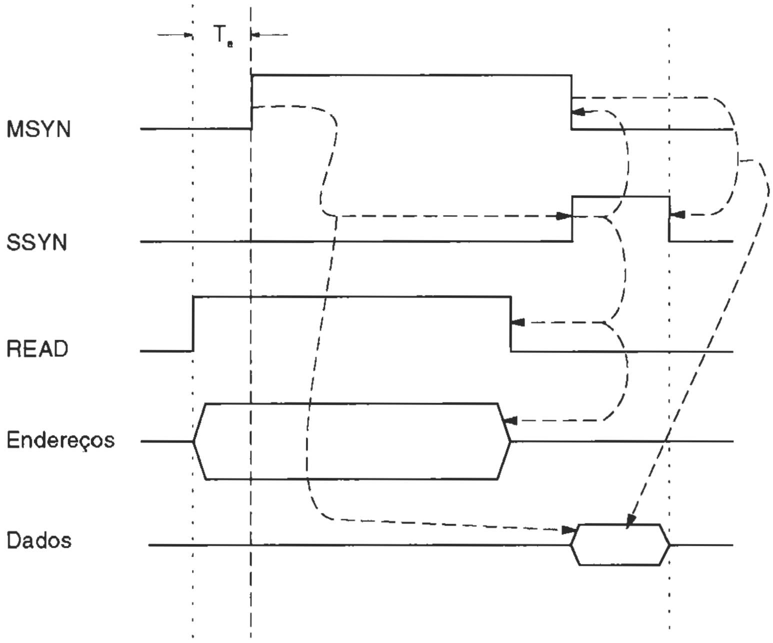
Com o método assíncrono, cada evento depende somente da ocorrência de evento anterior, o qual pode ter duração diferente em tempo. No item 6.6.3.1 mostraremos algumas diferenças entre os dois métodos, assim como suas vantagens e desvantagens.

Na Fig. 6.42 é apresentado o diagrama de tempo da mesma operação de leitura exemplificada para o caso de barramento síncrono, só que, neste caso, ela está sendo realizada para um *barramento assíncrono.*

Foram usadas as seguintes linhas para o exemplo em questão: de endereços, de dados, de READ, de iniciação do processo (MSYN — master synchronization ou sincronização de mestre) e de resposta do escravo (SSYN — slave synchronization ou sincronização de escravo). Quando a UCP deseja realizar uma operação de leitura, a UC coloca o endereço da célula de memória (se o acesso foi à memória, o que não é necessariamente verdade, pois poderia ser a um periférico), ativa o sinal de leitura — READ, para identificar qual a operação a ser realizada, e ativa o sinal MSYN, para indicar que a ação deve ser realizada, com o endereço informado. O sinal MSYN não é ativado junto com o de READ e da colocação dos bits de endereço no barramento; ele aparece depois, de modo a dar tempo aos sinais de endereço para se estabilizarem na linha.

Assim que o escravo detecta ("sensa") o sinal MSYN, ele inicia imediatamente a operação requerida, isto é, decodifica o endereço e coloca os dados no barramento. Ao concluir esta atividade, o escravo informa através da ativação de um sinal de resposta, SSYN. E o barramento volta a estar disponível para qualquer outra operação.

Como mostramos alguns diagramas de tempo, é interessante consolidar neste ponto algumas convenções comumente estabelecidas para a representação de informações nestes diagramas.



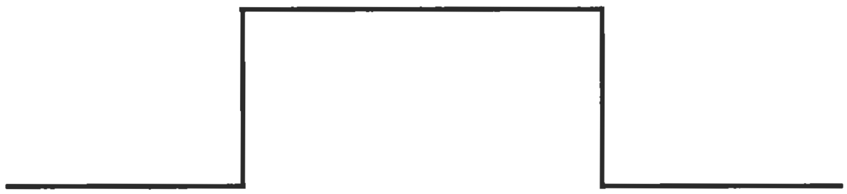
T_e- tempo de atraso entre endereços na linha e ativação de MSYN, para permitir estabilidade de endereço.

Figura 6.42 Operação de leitura em um barramento assíncrono.

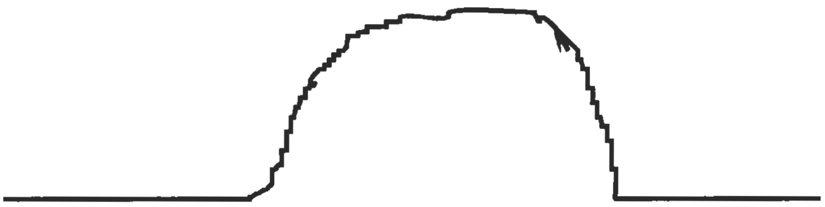
a) Os sinais que transitam nas linhas podem assumir dois níveis de tensão elétrica; um alto, correspondente ao bit 1, e outro, baixo, correspondente ao nível 0.

b) Linhas que transportam grupos de bits, como as de endereço ou de dados, podem ou não ser representadas como uma linha mais larga para indicar a noção de grupos de bits.

c) Em um sistema real, a transição do 0 para o 1, denominada borda frontal (*leading edge*), e a transição do 1 para o 0, chamada borda traseira (*trailing edge*), é realizada em um período de tempo finito (e não instantaneamente), o que acarretaria a representação do sinal como mostrado na Fig. 6.43(b). No entanto, costuma-se representar o sinal de forma mais quadrada, como se subida e descida fossem instantâneas (Fig. 6.43(a)).



(a) Sinal digital ideal



(b) Sinal digital real

Figura 6.43 Exemplo de formas ideal e real de um sinal digital.

d) Em um barramento assíncrono, tendo em vista que não há unidade fixa de tempo para relacionar as tarefas de uma dada operação, não há qualquer tipo de relação entre os vários sinais que circulam no barramento. Por essa razão, é comum o uso de linhas para unir o sinal que origina um outro (ou outros), terminando com uma seta na ponta do sinal que sucede.

No exemplo da Fig. 6.42, com os sinais iniciais, READ e de endereços, nada acontece, pois estão sendo ativados pela UC. Em seguida (após o tempo T_a), ocorre o sinal MSYN, que ativa as linhas de dados (observe a linha partindo da borda frontal do sinal MSYN e a seta na borda frontal da linha de dado) e faz o escravo ativar SSYN. A ativação do sinal SSYN (borda frontal) acarreta a desativação das linhas de endereço (seta na borda traseira), da linha READ e do sinal MSYN. Por sua vez, quando MSYN é desativado, SSYN também o é, e o ciclo de leitura é encerrado.

6.6.3.1 Comparação entre o Barramento Síncrono e Assíncrono

O *barramento síncrono* é simples de implementar e testar, justamente devido à sua natureza inflexível no tempo. Em conseqüência, qualquer atividade entre mestre e escravo somente pode se realizar em quantum fixo de tempo, o que se torna uma desvantagem. Isto porque o barramento síncrono pode ter problemas, por exemplo, ao trabalhar com dispositivos que tenham tempos de transferência diferentes. O que não acontece com o barramento assíncrono, que, por não depender de relógio com intervalos fixos de tempo, pode conviver com dispositivos que tenham velocidades baixa e alta, que utilizam tecnologia antiga e avançada. Assim, em um certo momento, a UCP pode operar com um determinado dispositivo (escravo), que opera com certa taxa de transferência (tempo). Os sinais se sucedem no barramento a partir da colocação do endereço, READ e MSYN na linha, independentemente de sua duração. Por isso, a operação seguinte pode ser realizada com outro dispositivo que tenha velocidade maior ou menor, porque tudo acontecerá também a partir do MSYN.

6.6.4 Tipos de Controle em um Processador

A UC é o elemento da UCP cuja função consiste em coordenar a execução completa de uma instrução de máquina (dos passos que caracterizam um ciclo de instrução), isto é, ela emite os sinais de controle para ativar a realização de cada etapa do ciclo da instrução (ver Fig. 6.20). Os sinais de controle para ativar as etapas são diferentes para cada instrução, pois as tarefas a serem executadas para a ação de somar são diferentes das da ação de subtrair.

A UC deve, então, ser construída contendo a programação de emissão destes sinais conforme a instrução que será decodificada.

A execução de um programa consiste, na realidade, na consecução de uma série de pequenos passos pelo hardware, menores talvez do que pudéssemos supor inicialmente. A Fig. 6.44 mostra um diagrama da decomposição de um programa em etapas cada vez menores, cuja efetiva realização vai nos auxiliar a entender melhor o funcionamento da área de controle e seus diferentes modos de implementação.

O diagrama mostrado na Fig. 6.44 contempla apenas as etapas de realização de um programa escrito em linguagem de máquina. Na realidade, programas costumam ser desenvolvidos em linguagens menos simples, redundando em comandos mais complexos, estando, pois, em um nível superior do diagrama (ver Cap. 9).

Há duas maneiras utilizadas no projeto e no funcionamento de uma UC de caracterizar conceitos diferentes de controle:

- controle programado diretamente no hardware (*hardwired control*); e
- controle por microprogramação.

A diferença básica entre os dois tipos está no processo de controle da realização do ciclo de instrução. No primeiro caso (*hardwired*), cada etapa é realizada segundo uma lógica preestabelecida, implementada fisicamente no hardware da área de controle. O item 6.6.4.1 descreve o processo. No caso de controle microprogramado (item 6.6.4.2), a interpretação e as conseqüentes etapas do ciclo de instrução são realizadas passo a passo por um programa, denominado microprograma.

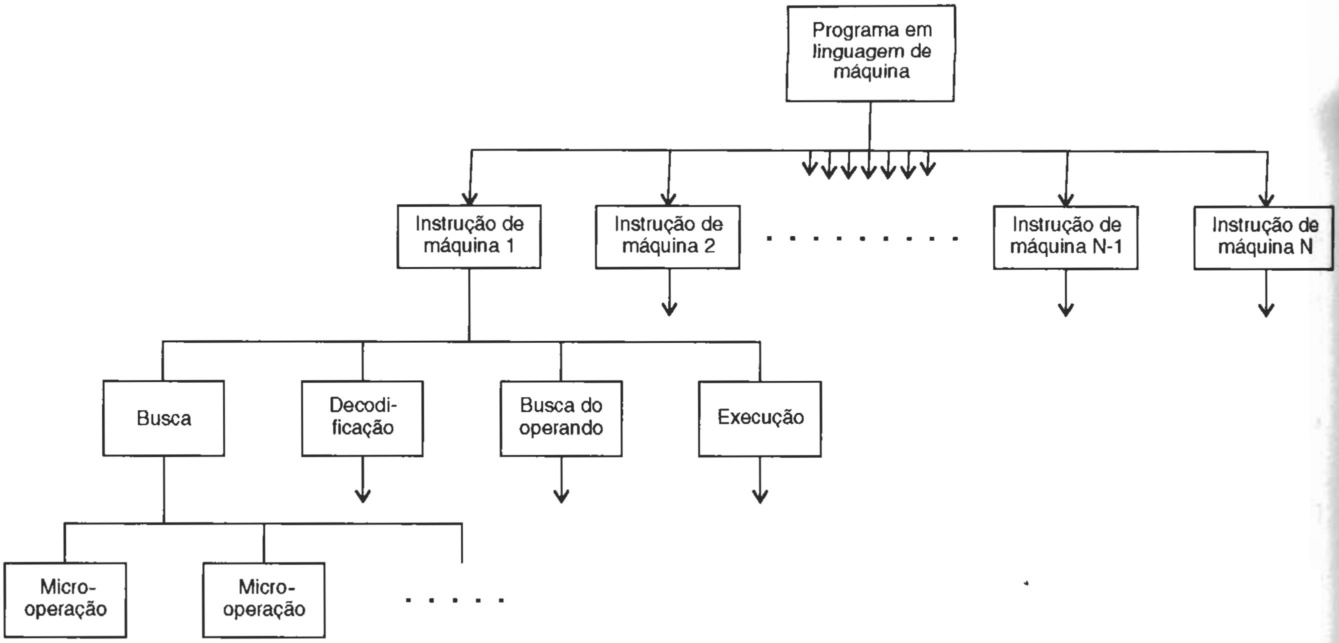


Figura 6.44 Decomposição de um programa em linguagem de máquina em seus elementos mais simples.

6.6.4.1 Controle Programado no Hardware

Neste tipo de implementação, a unidade de controle é construída como um conjunto de circuitos logicamente combinados (ver Cap. 4, para uma descrição sobre circuitos combinatórios), os quais produzem sinais de controle de saída de acordo com os sinais de entrada recebidos no circuito. As Figs. 6.15 e 6.16 mostram, no seu conjunto, as características de uma UC deste tipo, o que pode ser consolidado no esquema da Fig. 6.45, com todos os principais componentes da área de controle apresentados de forma integrada.

Para entender adequadamente como funciona este tipo de controle, vamos apresentar um exemplo prático, extraído de [STAL, 00] e [BUCH, 771].

Consideremos uma UCP simples, com um único ACC, uma UAL, o RI, o CI, o REM e o RDM e o barramento interno que conduz os sinais de informação entre eles, conforme mostrado na Fig. 6.46. A figura não mostra a fiação por onde passam os sinais de controle gerados na UC, mas, para auxiliar o nosso entendimento, ela apresenta os pontos terminais destes sinais, numerados como C, e com um círculo ao lado da correspondente numeração.

A unidade de controle recebe entrada do relógio (vamos ignorar o gerador de sinais, mostrado na Fig. 6.45), entrada do registrador de instrução (através do decodificador) e dos flags. Em cada ciclo de relógio, a unidade de controle realiza a leitura de todas as entradas, as quais passam por alguns de seus circuitos lógicos internos, e emite os correspondentes sinais de saída.

A movimentação de dados de um local para outro é um dos eventos mais freqüentes durante a realização de um ciclo de instrução. Anteriormente mencionamos que isto ocorre através da abertura de uma porta lógica no caminho entre os dois locais (ver Cap. 4) e que esta porta era aberta como conseqüência da chegada, em uma de suas entradas, de um sinal de controle.

O círculo C_4 da Fig. 6.46 mostra exatamente a porta lógica que permite, por exemplo, que uma instrução vinda da memória seja transferida do RDM para o RI, durante o ciclo de busca. Isto ocorrerá quando, em C, chegar um sinal de controle da UC.

A essência do processo consiste em entendermos como funcionam os circuitos combinatórios internos da UC. Para cada sinal de entrada, produz-se um sinal de saída derivado de uma expressão booleana implementada nos circuitos da UC.

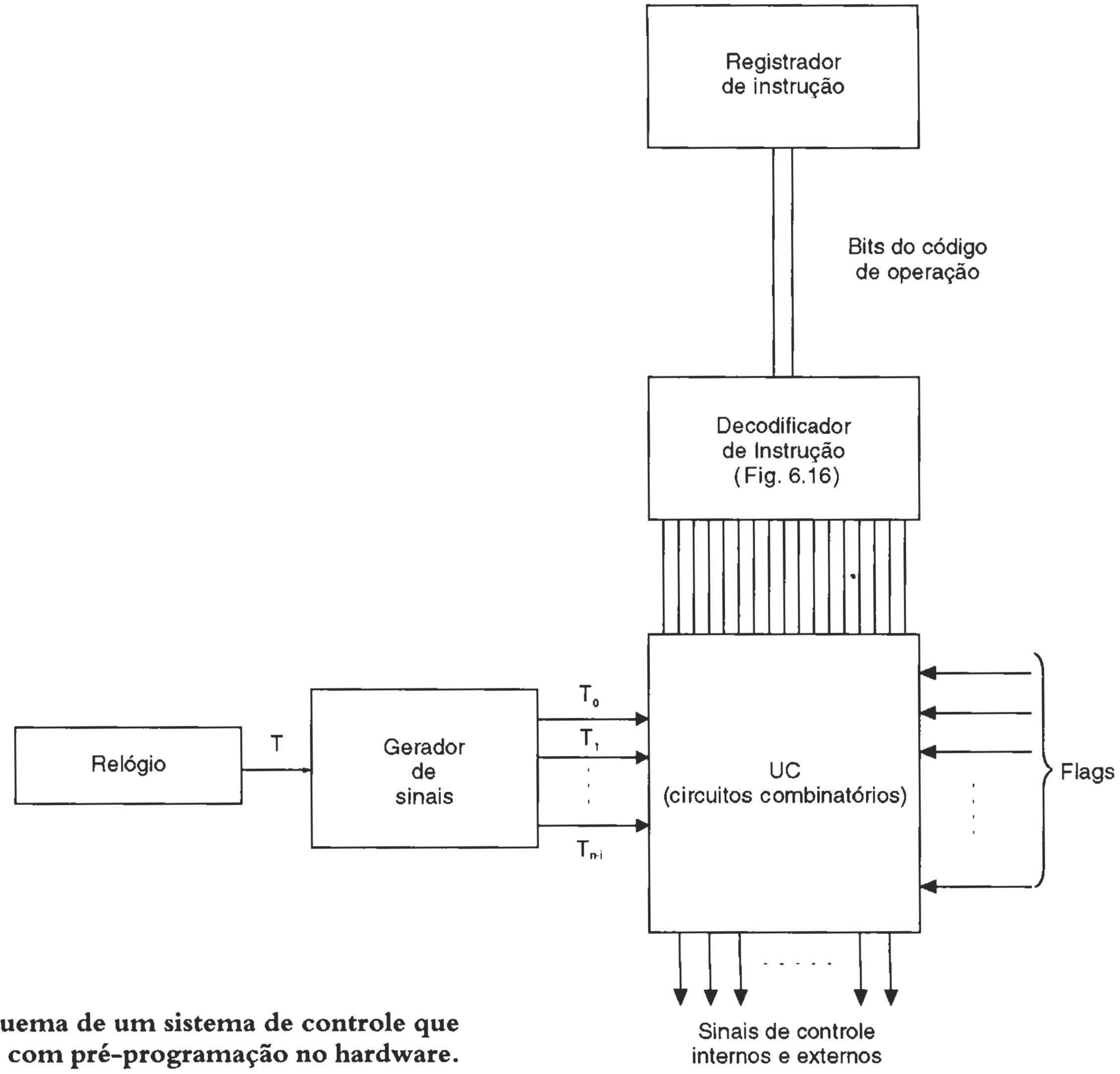

Registrador de instrução
Bits do código de operação
Decodificador de Instrução (Fig. 6.16)
Relógio
T
Gerador de sinais
T_0
T_1
T_{n-1}
UC (circuitos combinatórios)
Flags
Sinais de controle internos e externos

Figura 6.45 Esquema de um sistema de controle que se utiliza de UC com pré-programação no hardware.

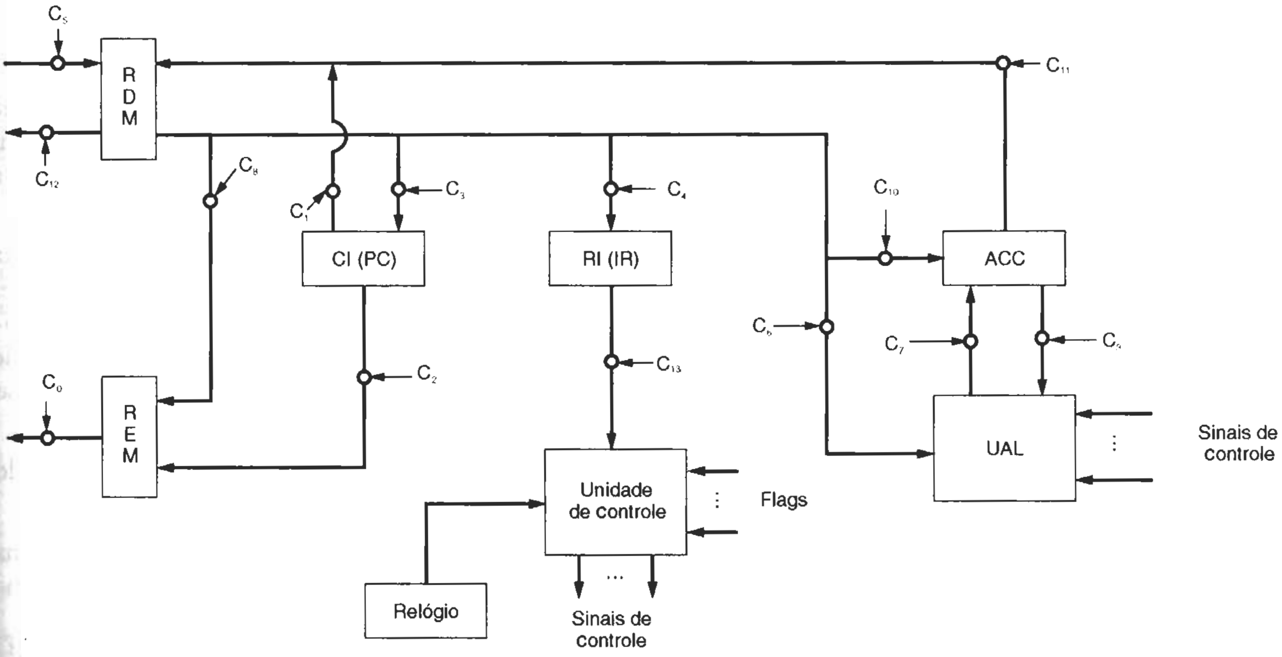

C_5
R D M
C_{12}
C_8
C_1
C_3
C_4
C_{11}
C_{10}
CI (PC)
RI (IR)
ACC
C_6
C_7
C_9
C_0
C_2
C_{13}
R E M
UAL
Sinais de controle
Unidade de controle
Flags
Relógio
Sinais de controle

Figura 6.46 Sinais de controle em um pequeno sistema.

Vamos considerar o sinal C_5 de controle. Este sinal acarreta a leitura de dados do barramento externo para o RDM. Além disso, no ciclo de busca de um ciclo de instrução temos:

t_1: REM ← (CI) sinal de controle ativo: C_2
t_2: (CI) ← (CI) + 1 sinais de controle ativo: C_5 e READ
RDM ← Memória
T_2: RI ← (RDM) sinal de controle ativo: C_4

É necessário ainda definir dois novos sinais de controle: P e Q, onde $P = 0$ e $Q = 0$ para o ciclo de busca. Desta forma, pode-se definir C_5 pela seguinte expressão booleana:

$C_5 = P.Q.T_2 + P.Q.T_2$ ou
$C_5 =$ (not P and not Q and T_2) or (not P and Q and T_2)

Isto é, o sinal C_5 será igual a 1 durante o 2.° período de tempo (T_2) no ciclo de busca ($P.Q.T_2$) e em outro estágio do ciclo de instrução não constante de nosso exemplo.

A equação lógica mostrada ainda não está completa. Falta incluir a parte relativa à execução de instruções. Suponhamos, por exemplo, que as instruções LDA e ADD são as únicas (do conjunto de instrução de nosso computador) que realizam leitura de dados da memória. Assim,

$C_5 = P.Q.T_2 + P.Q.T_2 + P.Q\,(LDA + ADD).T_2$

onde $P = 1$ e $Q = 0$ para o ciclo de execução e LDA/ADD são resultados da decodificação.

O processo exemplificado, que resultou na expressão lógica para C_5, deve ser repetido para cada um dos outros sinais de controle gerados pela UCP. A conseqüência final é a construção de inúmeras expressões lógicas que definem o comportamento da UC e, por fim, da UCP. Pelo exemplo da Fig. 6.46 podemos imaginar a enorme quantidade de expressões lógicas que devem ser definidas para uma UCP grande e real, o que garante extrema complexidade à tarefa. Além disso, há uma grande inflexibilidade no método, visto que qualquer alteração que se deseje fazer em algum elemento da UCP (inclusão de dispositivo, etc.) acarretará a necessidade de redefinição de todo o conjunto de expressões lógicas. A vantagem do processo é que a instrução de máquina buscada é imediatamente executada pelo hardware, com o conseqüente ganho de velocidade. Processadores com arquitetura RISC utilizam esta técnica de funcionamento da UC visando justamente a este ganho de velocidade na execução dos ciclos das instruções. Como as instruções de máquinas RISC tendem a ser mais simples, também a tarefa de especificar as inúmeras expressões lógicas tem sua complexidade um pouco reduzida.

6.6.4.2 Controle por Microprogramação

Conforme foi observado no item anterior, o funcionamento da unidade de controle de um processador já é bastante complexo por sua própria natureza (sincronização e geração de uma enorme quantidade de sinais para controlar, no tempo certo, a efetivação de várias ações dentro do processador), especialmente se este controle tiver que ser (como no item anterior) realizado através de um conjunto de circuitos lógicos.

Em que consiste, na realidade, a execução de um ciclo de instrução? Na realização, já vimos, de uma certa quantidade de pequenas operações, basicamente de transferência de valores binários entre registradores e, eventualmente, de uma operação matemática qualquer. Estas "pequenas" operações (pequenas pela sua simplicidade e curta duração) são denominadas *microoperações*. A Fig. 6.47 mostra um exemplo de um ciclo de busca (primeira parte do ciclo de uma instrução) contendo não só as microoperações a serem realizadas, mas também os períodos de tempo distintos entre elas.

Na Fig. 6.47(b) cada linha mostrada se constitui em uma microoperação; a linha em que não está mostrado o período **t** é aquela que se realiza no mesmo instante que a linha anterior.

Uma microoperação é a menor ação que pode ser realizada em um processador, consistindo em geral na ativação de um flip-flop (por um pulso de relógio), ou ainda na abertura de uma porta lógica para movimentação de dados de um registrador para outro.

Uma microinstrução é o processo prático de definir uma microoperação, determinando qual porta lógica deve ser aberta ou em qual registrador um sinal de relógio deve ser introduzido. Deste modo, para cada

RI → (CI)

(CI) → (CI) +1

(a) Ciclo de busca simplificada

t_1 : REM → (CI)
Sinal READ

t_2 : RDM → (M(REM))

t_3 : RI → (RDM)
(CI) → (CI) +1

(b) Ciclo de busca realizado em 3 períodos de tempo

Figura 6.47 Exemplo de um ciclo de busca decomposto em microoperações.

microoperação mostrada na Fig. 6.47 deve haver uma microinstrução que indique o sinal apropriado a ser emitido para sua realização. Por exemplo, para efetivar a transferência dos bits de endereço de uma instrução, os quais estão armazenados no CI (1.ª linha da Fig. 6.47(b)), para o REM, há uma porta lógica no caminho entre CI e REM (uma linha de controle, vindo da UC, une os dois), que será ativada (bit 1 aparece durante o intervalo de tempo correspondente a t_1) pela interpretação da microinstrução correspondente.

Para melhor entendimento do que será descrito a seguir, vamos considerar que macroinstrução (termo não utilizado normalmente, sendo adotado aqui em contraposição à microinstrução, este sim largamente difundido) é uma instrução em linguagem de máquina (mnemônico assembly do tipo ADD, LDA etc.), e que o processo de executar uma macroinstrução é denominado interpretação (não confundir com a técnica de interpretação de programas, conforme descrição no Cap. 9), se foi realizado, é claro, segundo a metodologia de microprogramação.

O termo microprogramação foi primeiramente utilizado por M.V. Wilkes (Wilkes era professor no Cambridge University Mathematical Laboratory), em 1951, em um artigo por ele divulgado, em que propunha que as ações da unidade de controle de um computador fossem realizadas através da execução de instruções (conceitualmente semelhantes a instruções em linguagem de máquina) que, como resultado, produziriam a realização de uma ação ou operação aritmética/lógica pelo hardware. Estas instruções foram denominadas por Wilkes de microinstruções, para diferençar da instrução de máquina (macroinstrução) que as originou e, também, porque realizam uma operação de mais curta duração que o tempo gasto para realizar uma macroinstrução. O conjunto de microinstruções foi, então, chamado de microprograma.

Assim, foi inserido um nível intermediário entre o nível de linguagem de máquina (que era até então o nível mais baixo de software) e o nível de hardware. Este nível, denominado firmware, é constituído pelo microprograma citado.

O primeiro e mais famoso sistema de grande porte microprogramado foi o sistema IBM/360, anunciado pela IBM em abril de 1964. Na prática, parece que temos uma micro-UCP no interior da UCP, visto que deveremos ter para executar o microprograma:

- Memória — usualmente chamada memória de controle, que armazena as microinstruções. É, em geral, do tipo não-volátil, isto é, ROM, de modo que um usuário não destrua acidental ou intencionalmente seu conteúdo.
- Microcontador de instruções — para armazenar o endereço da próxima microinstrução.
- Microrregistrador de instrução — que armazena a microinstrução correntemente sendo interpretada.

A Fig. 6.48 mostra um exemplo de unidade de controle microprogramada.

Tipos de projeto de microinstruções

Há dois métodos de formatar e usar uma microinstrução:

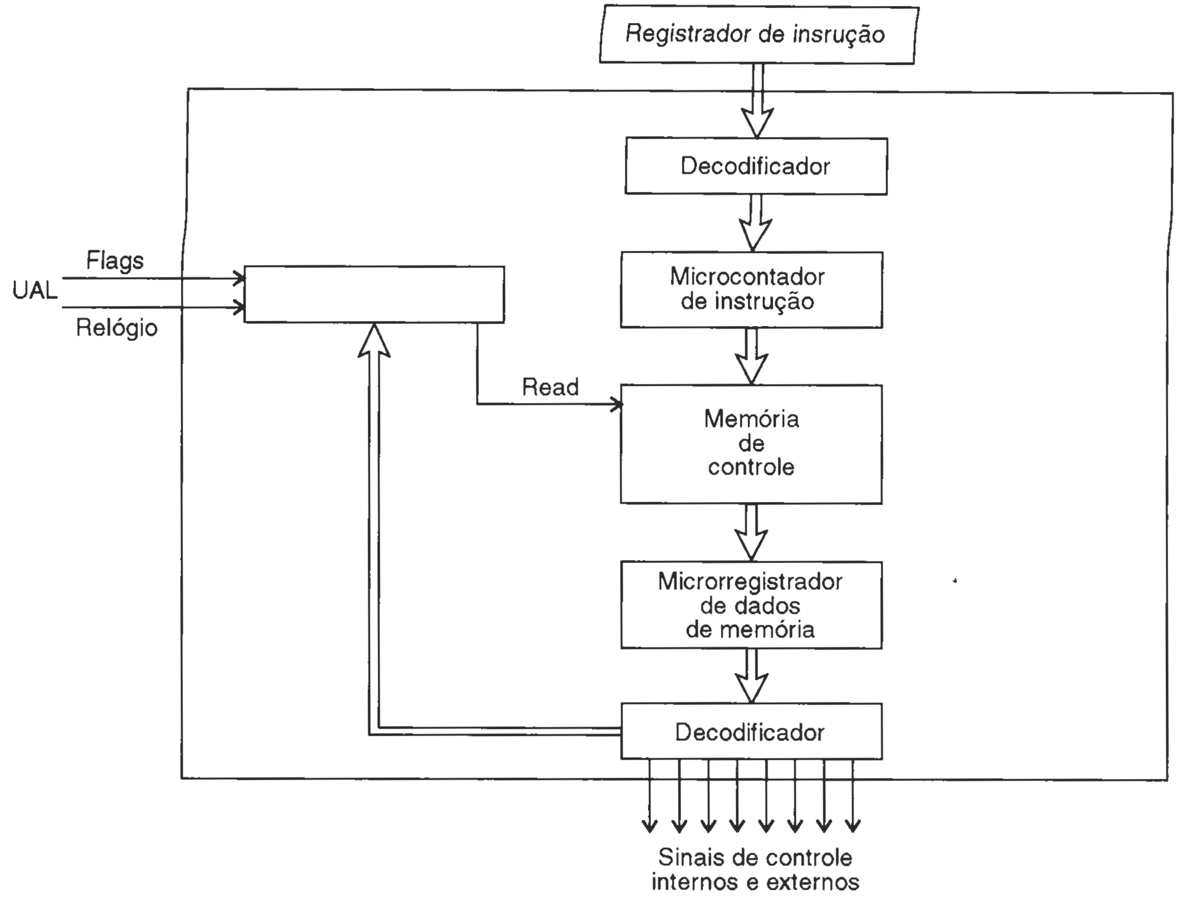

Registrador de insrução
Decodificador
Flags
UAL
Relógio
Microcontador de instrução
Read
Memória de controle
Microrregistrador de dados de memória
Decodificador
Sinais de controle internos e externos

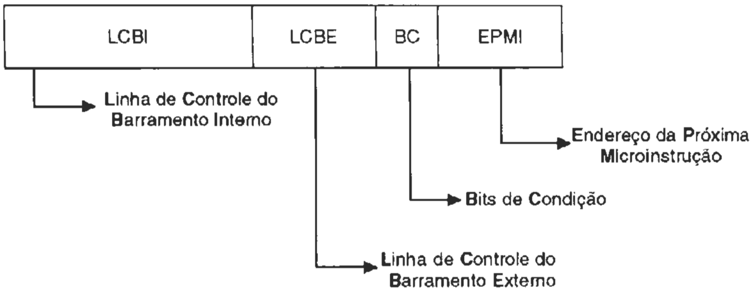
Figura 6.48 Exemplo de unidade de controle microprogramada, com microinstruções verticais (uso de decodificador).

- microinstruções horizontais; e
- microinstruções verticais.

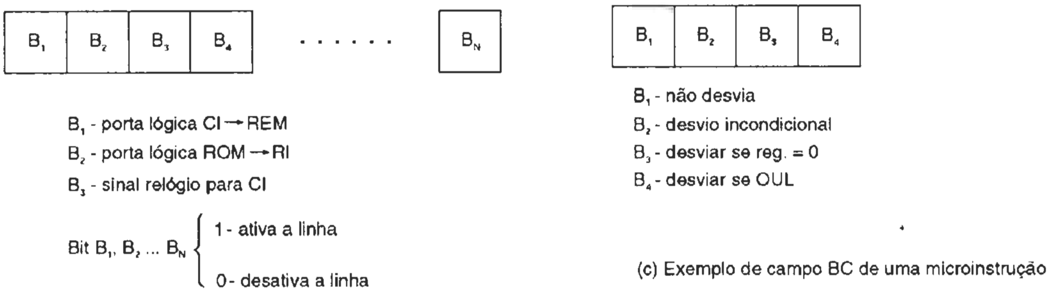
Uma microinstrução horizontal (ver Fig. 6.49) é projetada de modo que cada bit da microinstrução tenha uma função específica — controlar uma linha de controle interna da UCP (por exemplo, abrir uma porta lógica); controlar uma linha do barramento externo de controle; definir uma condição de desvio e endereço de desvio, quando for o caso. Uma instrução desse tipo é executada através:

- da ativação das linhas de controle cujo correspondente bit é de valor igual a 1 (a microoperação é realizada). As demais (valor do bit correspondente igual a zero) permanecem como anteriormente;
- da avaliação dos bits de condição: se todos forem iguais a zero, deve-se executar a próxima microinstrução em seqüência, isto é, não há desvio. Se um dos bits de condição for igual a 1, então será executada em seguida a microinstrução cujo endereço na memória de controle consta do campo de endereço (campo EPMI da Fig. 6.49).

Um formato deste tipo tem a vantagem de ser o mais simples e direto possível, podendo controlar várias microoperações em paralelo (no mesmo ciclo de relógio). Ele tem uma outra vantagem potencial, referente a uma eficiente utilização do hardware, pois pode iniciar várias microoperações simultâneas. No entanto, tem uma grande desvantagem no que se refere à ocupação de espaço de memória de controle. Justamente porque usa um bit para cada ação, e como em um computador de razoável capacidade de processamento a quantidade de microoperações é grande, as microinstruções têm um tamanho normalmente extenso (às vezes, mais de 60 bits são usados em uma microinstrução) e, em conseqüência, o tamanho requerido para a memória de controle também o é, o que acarreta elevação de custos para o projeto da UCP.


LCBI
LCBE
BC
EPMI
Linha de Controle do Barramento Interno
Endereço da Próxima Microinstrução
Bits de Condição
Linha de Controle do Barramento Externo

(a) Exemplo de um formato de microinstrução horizontal


B₁ B₂ B₃ B₄ ...... B_N
B₁ - porta lógica CI → REM
B₂ - porta lógica ROM → RI
B₃ - sinal relógio para CI
Bit B₁, B₂ ... B_N { 1 - ativa a linha / 0 - desativa a linha
B₁ B₂ B₃ B₄
B₁ - não desvia
B₂ - desvio incondicional
B₃ - desviar se reg. = 0
B₄ - desviar se OUL

(b) Exemplo de campo LCBI de uma microinstrução

(c) Exemplo de campo BC de uma microinstrução

Figura 6.49 Exemplo de microinstrução horizontal.

Para evitar o excesso de bits de uma microinstrução horizontal, pode-se introduzir um passo de decodificação no processamento da microinstrução. Em outras palavras, em vez de os bits da microinstrução acessarem diretamente uma linha de controle, esses bits podem significar o código de um grupo de ações. A unidade de controle necessitará, então, de um decodificador extra para identificar quais as linhas que serão efetivamente ativadas. Este tipo de microinstrução é denominado vertical. Assim, por exemplo, é possível incluir na microinstrução um código de 4 bits, que representa 16 linhas de controle. Desse modo, economizam-se 12 bits (16 − 4) na instrução. A Fig. 6.50 mostra um exemplo de formato de microinstrução vertical.

Um formato do tipo vertical tem a vantagem de reduzir o custo da UC, pois a memória de controle se torna menor em tamanho devido à redução dos bits das microinstruções, embora possa haver mais microinstruções. Por outro lado, ocorre uma perda de tempo devido à necessidade de decodificação dos campos de cada microinstrução e, nesse caso, o tempo de processamento da macroinstrução se torna maior do que quando se trata de microinstruções horizontais.

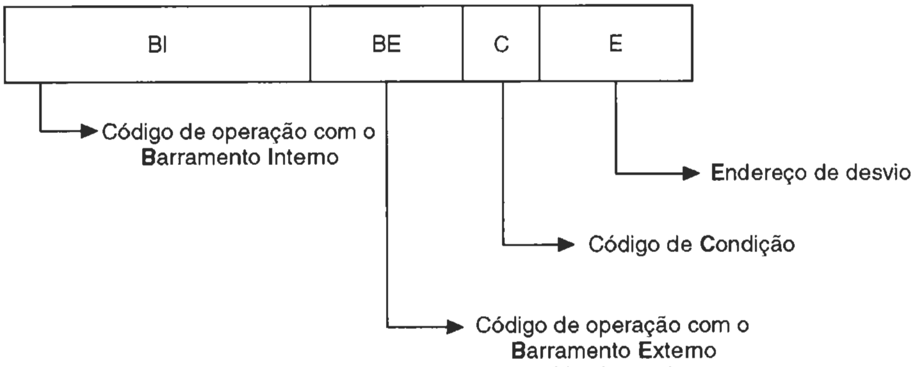

BI
BE
C
E
Código de operação com o Barramento Interno
Endereço de desvio
Código de Condição
Código de operação com o Barramento Externo (do sistema)

Figura 6.50 Exemplo de microinstrução vertical.

É possível criar um projeto de formato misto, de modo que a microinstrução resultante não seja totalmente *horizontal — isto é, com a maior quantidade possível de bits — nem totalmente vertical, isto é, com a codificação* de todos os campos. Este formato, conquanto reduza a quantidade de bits em relação ao formato horizontal (reduz o custo da memória de controle), não introduz acentuada perda de tempo em decodificação.

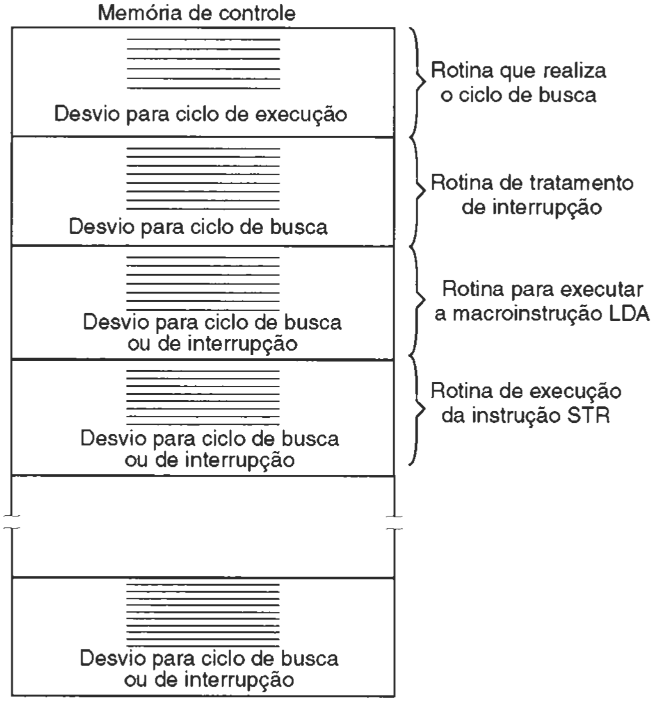

Memória de controle
Desvio para ciclo de execução
Desvio para ciclo de busca
Desvio para ciclo de busca ou de interrupção
Desvio para ciclo de busca ou de interrupção
Desvio para ciclo de busca ou de interrupção
Rotina que realiza o ciclo de busca
Rotina de tratamento de interrupção
Rotina para executar a macroinstrução LDA
Rotina de execução da instrução STR

Figura 6.51 Exemplo de configuração de memória de controle, contendo as diversas rotinas armazenadas.

Qualquer que seja a consideração, sempre haverá a diferença de desempenho e custo entre os dois formatos, induzindo os usuários a diferentes escolhas, de acordo com suas necessidades. O usuário que desejar um sistema de controle rápido, embora caro, certamente optará pelo formato horizontal, ao passo que o usuário cuja prioridade seja baixo custo, mesmo com perda de desempenho em tempo, deverá optar pelo modelo de microinstrução vertical.

Na realidade, um microprograma é constituído de várias microrrotinas (grupos de microinstruções com o mesmo propósito), seqüencialmente armazenadas na memória de controle, sendo que, em geral, a primeira rotina refere-se à realização do ciclo de busca de uma macroinstrução. A Fig. 6.51 mostra um exemplo de configuração de memória de controle com as diversas rotinas armazenadas em seqüência. O processamento normal consiste em um loop permanente com a 1.ª rotina (que realiza o ciclo de busca de uma macroinstrução) e, daí para diante, acessa-se a rotina referente à macroinstrução desejada, identificada pelo seu código de operação, através de desvio para o endereço apropriado, obtido durante a decodificação da C.Op.

6.6.5 Os Processadores e Suas Arquiteturas

6.6.5.1 A Evolução da Arquitetura x86 da Intel

Como já sabemos, o primeiro microprocessador efetivamente lançado pela Intel foi o 4004 (processador com palavra de 4 bits), e posteriormente a companhia cresceu consideravelmente no mercado com o lançamento do Intel 8080/8085, que veio a constituir-se no processador dos primeiros microcomputadores comerciais. A Fig. 6.52 mostra o diagrama em bloco do processador 8085.

Os processadores de 8 bits, anteriores à primeira geração da Arquitetura para PCs

Tanto o Intel 8080 quanto o 8085 possuíam palavra de 8 bits (Address Data Bus AD_0–AD_7), mesmo tamanho da palavra do processador Motorola 6800, lançado na mesma época. Os endereços tinham 16 bits, permitindo uma capacidade máxima de memória endereçável de 64 K células ou 64 Kbytes, visto que, cada célula podia armazenar 1 byte de dados (16 linhas, de AD_0–AD_7 e de A_8–A_{15}, no barramento de endereços — address bus).

O diagrama mostra o barramento interno de dados, de 8 bits, que interliga todos os elementos principais do processador, como a ALU — Unidade Aritmética e Lógica, o ACC — Acumulador, os registradores de dados, B, C, D, E, H, L, CI — Program Counter e Stack Pointer — Ponteiro da Pilha. Os demais elementos podem ser entendidos pelas explicações já apresentadas nos itens sobre a função processamento, a função controle e sobre as instruções de máquina.

Tratava-se de um processador que realizava o ciclo das instruções de forma seqüencial, possuindo 80 instruções de máquina (o Intel 8080 possuía 78 delas), sendo a maior parte de 8 bits (1 byte) de tamanho e outras de 2 e de 3 bytes.

Para economizar espaço na pinagem do chip, alguns dos pinos serviam a dois propósitos; assim, os 16 bits de endereços ficavam armazenados separadamente, sendo os 8 bits menos significativos, bem como os 8 bits de dados, fluindo pelo barramento de dados até o RDM (Address/Data Buffer) e os 8 restantes no REM (Address Buffer) via Address Bus.

A arquitetura x86 dos processadores para computadores pessoais (PCs), fabricados pela Intel, iniciou-se com o 8086 e logo depois o 8088, famoso por ter sido escolhido pela IBM para compor o computador pessoal

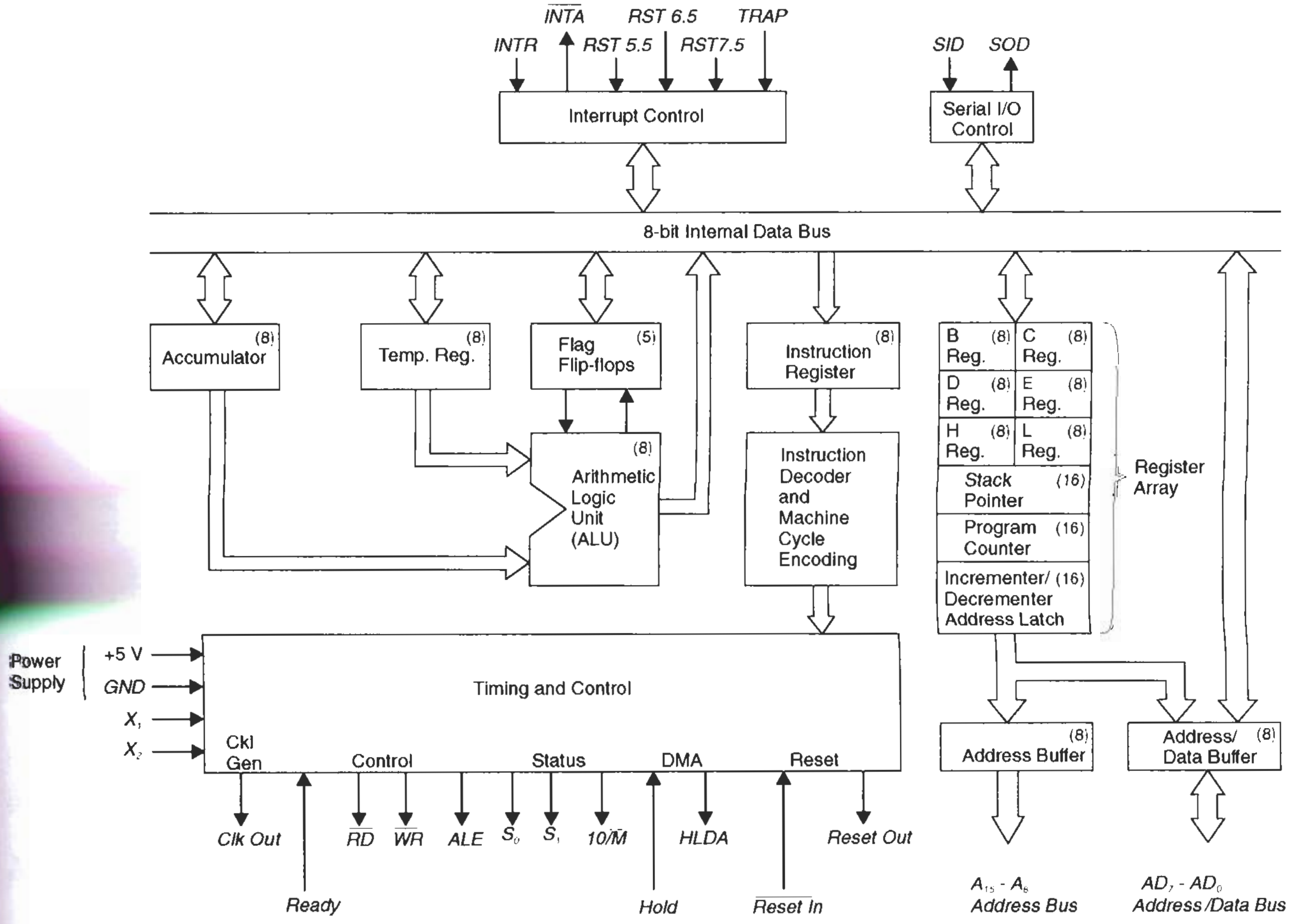

INTA
RST 6.5
TRAP
INTR
RST 5.5
RST7.5
SID
SOD
Interrupt Control
Serial I/O Control
8-bit Internal Data Bus
(8) Accumulator
(8) Temp. Reg.
Flag (5) Flip-flops
(8) Instruction Register
(8) Arithmetic Logic Unit (ALU)
Instruction Decoder and Machine Cycle Encoding
B (8) Reg.
C (8) Reg.
D (8) Reg.
E (8) Reg.
H (8) Reg.
L (8) Reg.
Stack (16) Pointer
Program (16) Counter
Incrementer/ (16) Decrementer Address Latch
Register Array
Power Supply
+5 V
GND
X_1
X_2
Timing and Control
Ckl Gen
Control
Status
DMA
Reset
Clk Out
RD
WR
ALE
S_0
S_1
10/M
HLDA
Reset Out
Ready
Hold
Reset In
(8) Address Buffer
Address/ (8) Data Buffer
A_{15} - A_8 Address Bus
AD_7 - AD_0 Address/Data Bus

Figura 6.52 Diagrama em bloco do processador Intel 8085.

IBM PC, que sacudiu o mercado de computadores pequenos, revolucionou aquela indústria e iniciou o caminho para o surgimento dos gigantes Microsoft, Compaq, Dell, Sun e outros fabricantes, sejam de software quanto de hardware para microcomputadores, praticamente eliminando do mercado as máquinas de grande e médio porte.

Os processadores Intel 8088/8086 constituíram a primeira geração da atual arquitetura concebida pela Intel e que, naturalmente, foi evoluindo ao longo do tempo até os dias atuais. A Fig. 6.53 apresenta o diagrama em bloco daqueles processadores.

A palavra daqueles processadores tinha 16 bits de tamanho e seu barramento de endereços tinha 20 bits de largura, de modo que podiam acessar até 1M células, cada uma podendo armazenar 1 byte de dados.

A diferença básica entre o 8086 e o 8088 é que o 8086 é essencialmente um processador completo de 16 bits, pois seus barramentos interno e externo permitiam a transferência de 16 bits, enquanto o barramento externo do 8088 permitia a passagem de apenas 8 bits de dados. Este último processador foi criado às pressas pela Intel, e esta redução de capacidade se deveu à exigência da IBM, que pretendia usá-lo nos IBM PC.

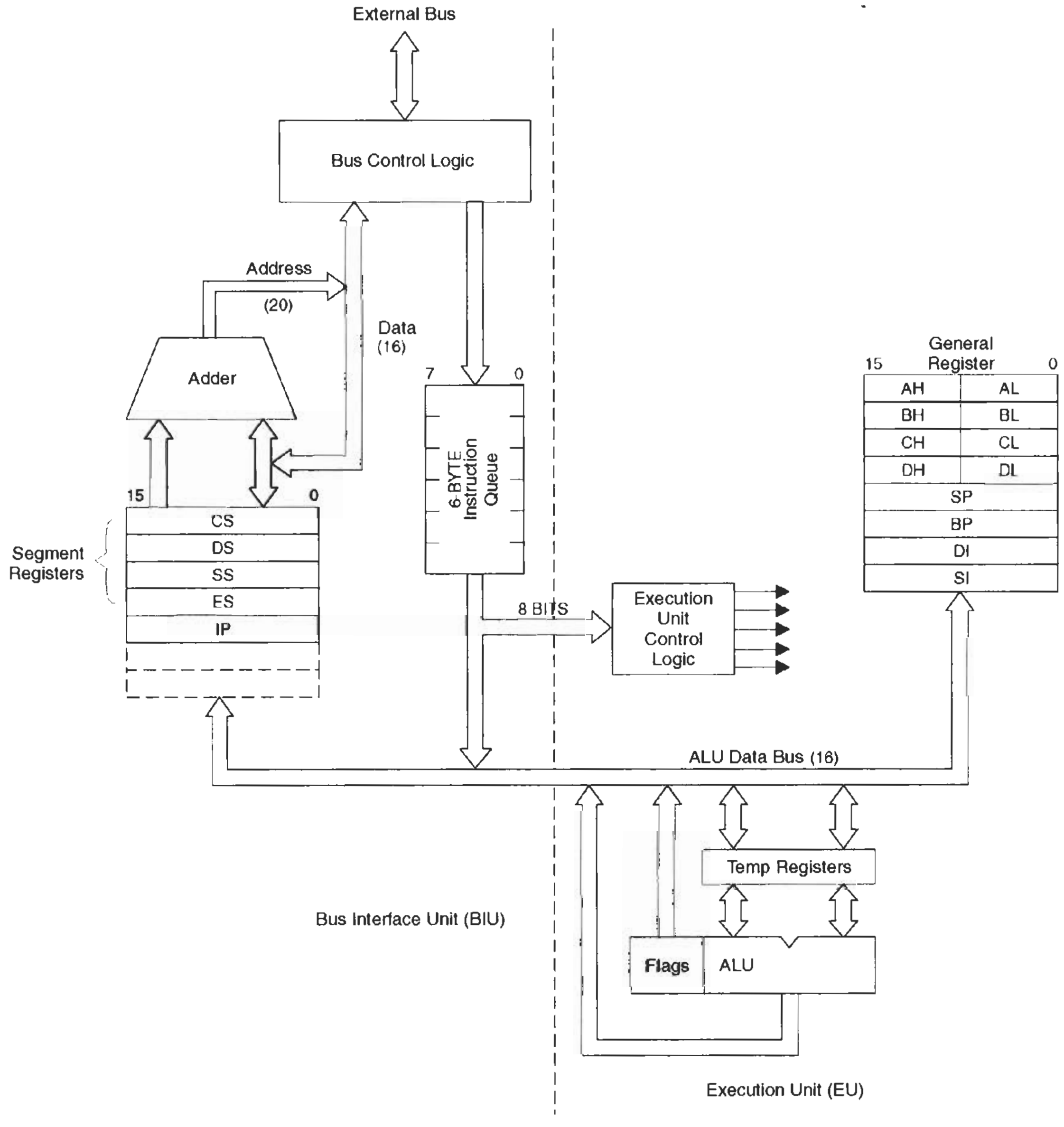

External Bus
Bus Control Logic
Address
(20)
Data
(16)
Adder
15
0
CS
DS
SS
ES
IP
Segment Registers
7
0
6-BYTE Instruction Queue
8 BITS
Execution Unit Control Logic
General Register
15
0
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
DI
SI
ALU Data Bus (16)
Temp Registers
Flags
ALU
Bus Interface Unit (BIU)
Execution Unit (EU)

Figura 6.53 Diagrama em bloco dos processadores Intel 8086/8088.

Como todos os periféricos na época somente transferiam 8 bits de cada vez, o 8086 teria problemas de interfaciamento.

Logicamente, o processador é dividido em duas unidades independentes, que se caracterizam como dois estágios de processamento *pipelining*: a unidade de execução (EU — Execution Unit) e a unidade de interligação com o barramento (BIU — Bus Interface Unit), estando na Fig. 6.53 separadas por uma linha tracejada. Ambas as unidades operavam de modo independente, aumentando o desempenho do sistema em relação aos 8080/8085.

A UE era constituída de 4 registradores com largura de 16 bits; eles tinham os nomes AX, que pode ser compreendido pela dupla AH/AL, cada um com 8 bits; BX, também dividido em BH/BL, o CX (CH/CL) e o DX (DH/DL). Como os registradores AH, AL, DH, etc. possuíam 8 bits de tamanho, isto os tornava compatíveis com o 8080/8085 e seus programas. Além disso, a UE compreendia a ALU — Unidade Aritmética e Lógica, os flags e registradores temporários (Temp Registers), a Unidade de Controle (Execution Unit Control Logic), sendo, assim, responsável pela decodificação do código de operação das instruções e da efetiva execução da operação indicada.

A unidade de interligação de barramento (BIU) processava todas as solicitações da UE, como ler e escrever dados com a memória e os dispositivos de E/S, sendo, também, responsável por toda a interação com o barramento do sistema (externo ao processador).

Uma grande diferença do 8088/8086 para o 8080/8085 era que os primeiros não possuíam um RI único, como esses, e sim uma unidade com capacidade de receber até 6 instruções de cada vez, denominada *prefetch queue* (fila de busca antecipada).

Embora os registradores de segmento possuíssem 16 bits de tamanho, os endereços no 8088/8086 tinham 20 bits, sendo o valor de 20 bits de efetivo endereço calculado da seguinte maneira (ver Fig. 6.54):

a) O valor do segmento usado é deslocado 4 vezes para a esquerda e, com isso, acrescentam-se 4 bits ao número que possuía 16 bits. Seja, por exemplo, o valor $357C_{16}$ ou 0011010101111100_2, que se transforma em 357C016 ou 00110101011111000000_2.

b) Ao valor 357C0 soma-se o valor armazenado no IP (instruction pointer, o nosso conhecido CI), por exemplo, 225F; com isso, obtém-se o endereço real, que é 37A1F, de 20 bits.

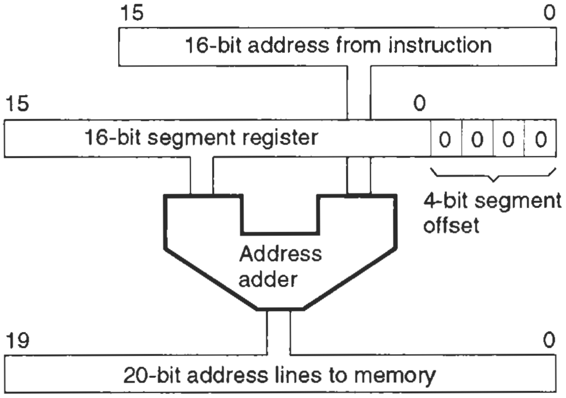


Figura 6.54 Cálculo de endereços nos processadores Intel 8086/8088.

Os processadores 8088/8086 funcionavam com uma freqüência de relógio de 4,77 MHz, tendo posteriormente sido lançadas versões operando com 8 MHz.

A segunda geração da arquitetura Intel para PCs — Intel 80286

O processador 286 foi utilizado nos computadores IBM AT, tendo um aumento apreciável de desempenho em relação aos 8088. Suas características principais eram:

Palavra: 16 bits.

Endereços: 24 bits, podendo acessar até 16 Mbytes de memória principal (cada célula também tinha 1 byte de tamanho).

Freqüência de relógio: de 6 MHz até 20 MHz.

Com o 286 a Intel introduziu o modo protegido em sua arquitetura, o que permitia que o sistema operacional protegesse programas aplicativos uns dos outros e dos programas dos usuários. Permitia, também, que o código do sistema operacional estivesse protegido dos demais programas. Como naquela época (1982/85) o DOS ainda era o sistema operacional dominante, e ele era um SO monousuário, o modo protegido praticamente não teve utilidade neste processador.

A arquitetura interna do 286 apresentou diversos aperfeiçoamentos em relação à do 8086/8088. A principal delas foi a existência de duas filas, de código e de instruções, além de o 286 possuir 4 unidades operacionais independentes, em vez das duas do 8086/8088. Elas eram: unidade de barramento (bus unit), unidade de instruções (instruction unit), unidade de endereçamento (address unit) e unidade de execução (execution unit). As duas primeiras unidades eram interligadas pela fila de código e a unidade de instrução era ligada à de execução pela fila de instruções.

'82 ©
N80286 - 12

Figura 6.55 Microprocessador 80286.

A unidade de barramento buscava instruções na memória e as encaminhava para a unidade de instruções. Esta informava a unidade de endereços dados para o cálculo dos endereços de acesso. A inclusão de uma unidade separada para calcular previamente endereços acelerou bastante o desempenho do 286 em relação ao 8086/8088.

A Fig. 6.55 mostra a pastilha de um processador 80286, no formato DIP, tradicional para a época, e a Tabela 6.1 contém mais detalhes da especificação do processador.

A terceira geração da Arquitetura Intel para PCs — Intel 80386

Com os processadores 386 a Intel lançou, em 1985, o primeiro time de 32 bits de palavra. O processador e demais membros da família 386 tiveram diversos aperfeiçoamentos substanciais. Dentro da nomenclatura e da arquitetura básica 386 surgiram diversos modelos, lançados não somente pela Intel mas também por novas empresas fabricantes de processadores surgidas na época, como a AMD e Cyrix.

Os processadores 386 Intel foram: 386SX, 386DX, 386SL. O DX foi o primeiro modelo da família lançado pela Intel, apresentando todas as características de uma máquina de 32 bits. Três anos mais tarde, com o propósito de afastar o uso do 286, ainda bastante empregado no mercado, a Intel lançou o modelo 386SX que, apesar de possuir palavra de 32 bits, tinha mais características do 286, como barramento externo de dados de 16 bits, endereços de 24 bits (acesso até 16MB de MP) do que do 386.

O processador 386DX tinha as seguintes características principais:

- palavra: 32 bits;

- endereços: 32 bits (capacidade de acessar até 4G células, todas de 1 byte de tamanho);
- freqüência de operação: de 16 MHz até 33 MHz;
- estágios de pipeline: 6 Bus Interface Unit-BIU, acessa a memória principal e os dispositivos de E/S para ler e escrever dados; Code Prefetch Unit, recebe as instruções de máquina do BIU e as coloca em uma fila de 16 bytes de tamanho; Instruction Decode Unit, decodifica o código de operação da instrução armazenada na Code Prefetch Unit e acessa o microcódigo referente; Execution Unit, executa as microinstruções; Segment Unit, executa o cálculo dos endereços, fazendo a soma do endereço de segmento e relativo do local de acesso e Paging Unit, transforma os endereços calculados em endereços físicos e reais de MP. Na verdade, nem sempre todos os estágios funcionavam em paralelo, exceto os quatro primeiros;
- co-processador matemático (80387) operando juntamente com o DX.

Já o processador 386SX possuía características menos avançadas, entre as quais podem-se citar:

- barramento externo de 16 bits, enquanto o interno continuou sendo de 32 bits;
- ausência de co-processador matemático (80387) nas placas-mãe;
- endereços de 24 bits (até 16 Mbytes de MP).

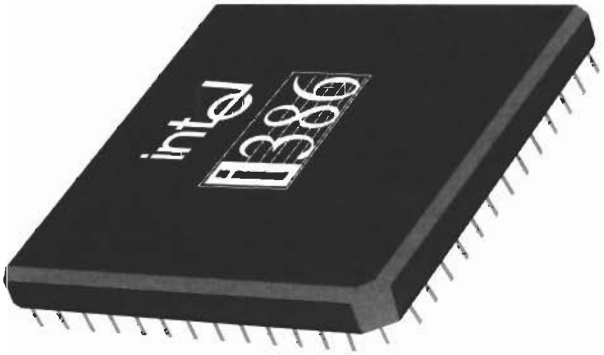


Figura 6.56 Microprocessador 80386.

Além do 386DX e 386SX, a Intel também lançou um processador mais voltado para computadores portáteis, o 386SL, que introduziu detalhes de economia de energia e para uso do sistema com baterias (voltagem de operação de 3,3 V), embora com freqüências de operação até 25 MHz.

Conforme já mencionado anteriormente, a Tabela 6.1 apresenta mais detalhes das características do 80386, cujo formato físico é mostrado na Fig. 6.56.

A quarta geração da Arquitetura Intel para PCs — Intel 80486

Em 1989 a Intel lançou o processador 80486, denominado 486DX, também com arquitetura de 32 bits, endereçamento de 4 Gbytes (32 bits de endereçamento) e diversos outros aperfeiçoamentos em relação à família 386, que melhoraram substancialmente seu desempenho. A Fig. 6.57 apresenta o diagrama em bloco dos processadores 486.

Os processadores 486DX são capazes de ter um desempenho 100% maior que os 386DX de mesma freqüência de operação, sendo ainda encontrados em sistemas mais simples, como para processamento de texto, acessos simples à Internet e jogos mais antigos.

O aumento do desempenho do 486DX em relação ao 386DX não ocorreu por aumento da largura dos barramentos de dados, interno ou externo, nem da largura da palavra e registradores internos de dados, mas sim por um conjunto de alterações internas, tais como:

- *aumento da velocidade de execução do ciclo das instruções;*
- aumento da quantidade de estágios de *pipelining* (são nove diferentes estágios que permitem até 5 instruções diferentes executadas simultaneamente);
- incorporação ao processador de uma quantidade de memória cache, denominada cache interna ou de nível L1. São 8 Kbytes de cache, que aumentam significativamente o desempenho do processador por evitar muitos acessos à MP;
- inclusão de uma unidade de operação em ponto flutuante no interior da pastilha do processador, em vez de empregar processador separado (co-processador matemático, como nos processadores anteriores);
- melhoria do funcionamento das placas-mãe em relação aos modelos anteriores.

Como em todos os modelos anteriores, o 486 permite que programas executáveis gerados por processadores anteriores possam ser nele executados sem maiores problemas.

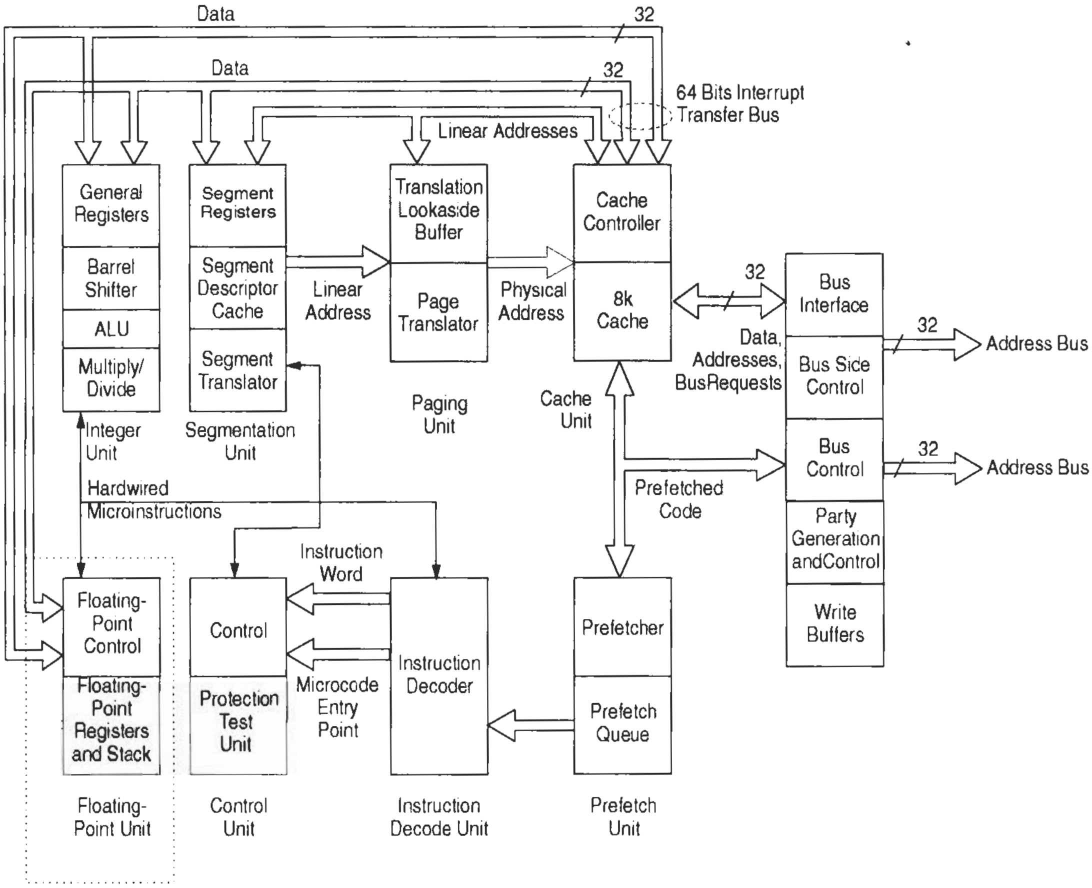

Data
32
Data
32
64 Bits Interrupt Transfer Bus
Linear Addresses
General Registers
Barrel Shifter
ALU
Multiply/ Divide
Integer Unit
Segment Registers
Segment Descriptor Cache
Segment Translator
Segmentation Unit
Linear Address
Translation Lookaside Buffer
Page Translator
Paging Unit
Physical Address
Cache Controller
8k Cache
Cache Unit
32
Data, Addresses, BusRequests
Bus Interface
Bus Side Control
Bus Control
Party Generation andControl
Write Buffers
32
Address Bus
32
Address Bus
Hardwired Microinstructions
Prefetched Code
Instruction Word
Floating-Point Control
Floating-Point Registers and Stack
Floating-Point Unit
Control
Protection Test Unit
Control Unit
Microcode Entry Point
Instruction Decoder
Instruction Decode Unit
Prefetcher
Prefetch Queue
Prefetch Unit

Figura 6.57 Diagrama em bloco do processador Intel 80486.

O diagrama em bloco da Fig. 6.57 mostra as seguintes unidades:

- *BIU — Bus Interface Unit* — com funções semelhantes à da BIU já descrita anteriormente; ela lê 16 bytes de dados de cada vez, do barramento externo para a cache interna, L1.
- *Cache Unit* — constituída de 8 Kbytes de memória e dos circuitos apropriados de controle. A cache do 486 funciona com o método associativo por conjunto (conjunto de 4) e política de escrita *write-through* (ver Cap. 5).

- *Code Prefetch Unit* — solicita à BIU novas instruções, as quais vão se enfileirando; a fila pode armazenar até 32 bytes e a quantidade de instruções depende do tamanho de cada uma.
- *Instruction Decode Unit* — lê uma instrução da fila e converte os bits de código de operação em sinais de controle para outras unidades e para o ponto de início do microprograma.
- *Control Unit* — contém o microprograma que faz a UCP funcionar, executando a instrução decodificada. Aciona as unidades de processamento de inteiros e de ponto flutuante.
- *Integer Unit* — responsável pela realização de operações aritméticas com valores representados em ponto fixo (inteiros). Para tanto, possui uma UAL, 8 registradores de 32 bits e um deslocador de bits para operações de multiplicação. Os dois barramentos de dados de 32 bits cada um (ver Fig. 6.57) servem para transferir 64 bits de cada vez.
- *Segmentation Unit* e *Page Unit* — juntas, essas unidades constituem a unidade de gerenciamento de memória do processador (MMU — memory management unit), empregada para implementar a gerência de proteção de memória e memória virtual.
- *Floating Point Unit* — responsável pela realização de operações aritméticas com valores representados em ponto flutuante, sendo em tudo semelhante aos co-processadores externos anteriores, 8087, 287 e 387, exceto que com melhor desempenho devido à sua integração ao processador.

A família 486 foi constituída de muito mais processadores que as anteriores. Diferentemente do 386, que teve lançados apenas o SX, DX e SL, os 486 foram lançados em mais versões, a saber:

486DX — já apresentado anteriormente, com palavra e barramento de 32 bits e freqüência de operação de 25 a 50 MHz.

486SX — uma versão apenas diferente do DX no que se refere à unidade de processamento de ponto flutuante, não existente no 486SX (na realidade, inibida neste processador). No entanto, os barramentos, interno e externo, e a palavra continuaram de mesmo tamanho no SX e no DX (diferentemente das diferenças entre o 386DX e 386SX). Além disso, o SX funcionava com freqüências de relógio mais baixas que no DX (de 25 MHz a 33 MHz no SX e de 25 a 50 MHz no DX).

486DX2 — este processador foi a primeira pastilha lançada pela Intel capaz de usar a tecnologia que permite ao processador funcionar no dobro da velocidade do barramento de memória. Assim, nestes processadores as velocidades eram 25/50 e 33/66 MHz. É claro que o aumento da velocidade do processador aumenta o desempenho do sistema, mas à medida que a sua velocidade aumenta mais, o desempenho não cresce tanto devido ao fato de que a memória começa a se tornar lenta em relação ao processador (acarretando o célebre *wait state* — estado de espera, onde o processador tem que esperar pela memória). Excetuando o aumento da velocidade, os DX2 são em tudo semelhantes aos DX, existindo ainda muitos desses processadores em sistemas no mercado.

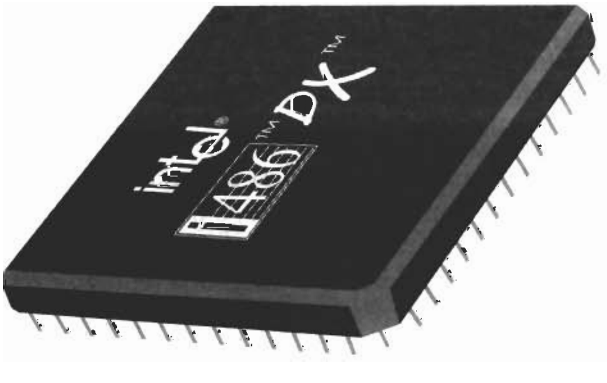


Figura 6.58 Microprocessador 80486DX.

486DX4 — este processador manteve a característica iniciada pela Intel com os 486DX2 de aumentar a velocidade do processador em uso com placas-mãe de menor velocidade. No caso, a velocidade do processador foi triplicada (e não quadruplicada como a sigla DX4 sugere). Assim, para placas de 25 MHz surgiu o DX4 de 75 MHz e para as placas de 33 MHz surgiram os DX4 de 100 MHz, que tiveram muito maior aceitação que o modelo anteriormente citado. Além dessa modificação, os DX4 também tiveram um aperfeiçoamento no tamanho da cache interna, L1, que dobrou de tamanho, passando dos 8KB dos modelos anteriores para 16KB.

A Tabela 6.1 mostra as características principais desses processadores e a Fig. 6.58 apresenta o aspecto físico da pastilha de um 486DX.

A quinta geração da Arquitetura Intel para PCs — Intel P5 — Pentium

A geração de processadores Pentium ou P5 apresentou um conjunto de modificações que alteraram a arquitetura até então vigente e aumentaram bastante o desempenho dos sistemas, como detalhado a seguir. O diagrama em blocos básico desse tipo de processador está mostrado na Fig. 6.5, enquanto a Fig. 6.59 mostra uma pastilha do Pentium original ainda no tipo de soquete DIP, formato quadrado, como nos processadores da família x86 até então. No conjunto, os processadores Pentium possuem um desempenho que se assemelha ao dobro da velocidade de um 486, considerando-se ambos com a mesma velocidade. Ou seja, diante de um Pentium e um 486 operando em 66 MHz, parece que o Pentium está na realidade operando em cerca de 120 MHz.

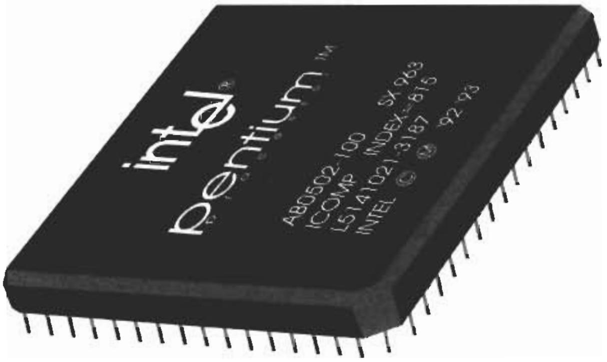


Figura 6.59 Microprocessador Pentium.

Antes de mostrarmos as características do Pentium original (ele poderia ser denominado Pentium I, em face do surgimento posterior do Pentium II e Pentium III), deve-se mencionar que o nome originalmente imaginado para o modelo era 586. No entanto, com o aparecimento no mercado das empresas AMD e Cyrix e sua conseqüente ameaça na concorrência, inclusive clonando anteriormente os modelos 486 que não podiam ser patenteados (porque números de modelos não podem ser patenteados), a Intel constatou que o processo de clonagem poderia continuar no 586 sem que ela pudesse impedir, a não ser utilizando um nome (e não número) que podia, então, ser patenteado, impedindo sua simples cópia por eventuais concorrentes.

Entre as características novas surgidas na arquitetura do Pentium original e que melhoraram seu desempenho global podemos citar (a Tabela 6.1 mostra mais detalhes dessas características):

- *Arquitetura superescalar* — ele funciona (ver a Fig. 6.5) com duas unidades de processamento de inteiros (por isso o nome superescalar) trabalhando em paralelo, o que pode se assemelhar a dois 486 em paralelo. As duas linhas pipelining desta arquitetura permitem, assim, a possibilidade de duas instruções poderem ser executadas em um único ciclo do relógio, se o programa (código executável) estiver preparado para usar essa vantagem, o que nem sempre ocorre.
- *Barramento de dados de 64 bits* — sendo o dobro do tamanho do barramento externo dos 486, este fato garante uma taxa de transferência entre UCP/MP maior que nos processadores anteriores.

- *Freqüência de operação do barramento de memória mais rápida* — nos Pentium essa freqüência pode ser de 60 MHz e 66 MHz, diferentemente dos 33 MHz dos 486. Além disso, os pentium estão preparados para operar com o barramento PCI (na época estavam novos no mercado).
- *Previsão de instrução de desvio* — no processo pipeline, pode acontecer um problema (ver item 6.6.2) no caso de instruções de desvio condicional. Ou seja, somente depois do teste de condição (etapa execução da operação) é que o sistema saberá que nova instrução deve ser buscada, se aquela referente ao teste ser verdade ou se aquela referente ao teste ser falso. Esta indefinição atrasa o processador. Para evitar isso, a Intel modificou o microcódigo de controle da cache, de modo que o processador busca as duas seqüências de instruções (do caso do teste resultar verdade e do caso do teste resultar falso), e, após o teste, o processador já terá disponível a instrução desejada, seja o resultado do teste verdadeiro ou falso.
- *Aperfeiçoamento na memória cache interna, L1* — a cache interna, L1, foi aumentada para 16 KB e dividida em duas partes separadas, cada uma com 8KB, sendo uma para dados e outra para instruções, como se pode observar no diagrama da Fig. 6.5. Além disso, a política de escrita foi alterada para *write back* em vez de *write through*, do 486.
- *Aumento de velocidade de operação* — os Pentium originais foram lançados ao longo do tempo com velocidades desde a inicial de 60 e 66 MHz até 200 MHz, sendo posteriormente substituídos pelos Pentium com instruções MMX (multimedia extension), que passaram a dominar o mercado.

Outros aperfeiçoamentos também podem ser citados como o aumento de desempenho da unidade de processamento de ponto flutuante (FPU) que, segundo a Intel, pode operar com um desempenho bem maior que a dos 486 e também a possibilidade de o sistema funcionar com dois processadores instalados na placa-mãe (se esta, naturalmente, tiver sido projetada para receber e funcionar com os dois processadores).

O Pentium teve um lado de fama negativa (além de suas notórias vantagens e desempenho superior) com o conhecido bug (erro) do processamento em ponto flutuante, que apareceu em um lote daqueles processadores, sendo posteriormente corrigido pela Intel, não sem antes ter sofrido um enorme desgaste devido mais à maneira como a empresa conduziu o assunto do que propriamente ao erro em si, não tão grave para a maioria dos seus usuários.

Em rápidas palavras e apenas para efeitos históricos, o problema e seus desdobramentos:

a) O problema, denominado FDIV, consistia no fato da perda de precisão em certas operações de divisão, utilizando-se instruções de ponto flutuante. Se você suspeitar que seu processador possa ter este bug (o que não parece provável devido ao longo tempo decorrido desde então — 1994), use uma planilha ou o calculador do sistema e divida o número 4.195.835 por 3.145.727, obtendo um determinado quociente. Em seguida, multiplique o resultado encontrado pelo mesmo divisor, 3.145.727, o que deve, naturalmente, resultar no mesmo valor inicial de dividendo, 4.195.835. Em um computador com o bug FDIV o resultado será 4.195.579, um erro de 256. Embora este teste possa não ser necessariamente conclusivo, pois alguns sistemas operacionais podem ter incluído uma correção, você poderá consultar a página Web da Intel a respeito.

b) O problema maior com este bug (muitos processadores são lançados com pequenos bugs, sem que o mercado seja basicamente afetado) foi a atitude adotada pelo fabricante que, inicialmente, não concordou com o problema e com a possibilidade de substituição incondicional.

c) Posteriormente, a Intel concordou em substituir todos os processadores defeituosos sem condição e sem ônus para os usuários, encerrando o episódio.

Ainda dentro da quinta geração de processadores, a Intel lançou em 1997 uma nova versão de seu Pentium, que acrescentava novas e inusitadas instruções, resultando em um tipo de processador denominado Pentium MMX. Este processador apresentou diversos aperfeiçoamentos sobre o Pentium original, substituindo-o definitivamente no mercado. O termo MMX, de multimedia extension, indica que o alvo dessas novas instruções são os aplicativos gráficos, que consomem muitas operações aritméticas com dados de poucos bits (em geral 8) como é o caso do valor de um pixel (ponto gráfico) a ser representado nos vídeos.

As instruções do tipo MMX (os Pentium foram lançados com 57 dessas instruções) permitem que uma instrução única possa realizar uma operação aritmética com vários pares de operando simultaneamente, desde que eles sejam pequenos, de modo que muitos cabem em um único registrador de ponto flutuante. É o caso, por exemplo, de dados de 8 bits, oito dos quais podem ser utilizados simultaneamente em um registrador de 64 bits.

Além disso, os processadores Pentium MMX surgiram com outras modificações em sua arquitetura, aumentando o desempenho global. Entre elas temos:

a) Aumento do tamanho da memória cache interna, L1 — passou de 16KB para 32KB, sendo 16KB para dados e 16KB para instruções.

b) Aumento do desempenho do mapeamento associativo por conjuntos da cache, de 2 para 4 conjuntos.

c) Aperfeiçoamento na unidade de previsão de desvios.

d) Aperfeiçoamento na unidade de decodificação.

A sexta geração da Arquitetura Intel para PCs — Intel P6 — Pentium Pro e Pentium II (Klamath, Deschutes e Celeron)

A *sexta geração de processadores* da Intel, P6, é constituída basicamente dos processadores Pentium Pro, lançado em novembro de 1995, do Pentium II lançado em maio de 1997, o qual teve alguns modelos subseqüentes, o Deschutes e o Celeron e do mais recente Pentium III. Em conjunto, esta família, a P6, possui características e aperfeiçoamentos em relação à geração anterior, a P5, especialmente as seguintes:

- mais uma unidade de processamento,
- melhor previsão para desvios e
- execução especulativa,

denominadas em conjunto pela Intel de *Dynamic Execution Microarchitecture*. No entanto, nesse caso, a distinção entre as gerações não é tão acentuada visto que até a mesma placa-mãe eles podem compartilhar.

O Pentium Pro foi o primeiro dos processadores de sexta geração, lançado pela Intel com o principal propósito de ser utilizado com melhor desempenho em servidores de rede. A Fig. 6.60 mostra uma pastilha do processador Pentium Pro (Fig. 6.60(a)) e um detalhe dessa pastilha, podendo-se observar a separação entre a cache L2 e o processador propriamente dito (Fig. 6.60(b)).

Entre os aperfeiçoamentos introduzidos na arquitetura do Pentium Pro, podem-se ressaltar:

- superpipelining — foi incluída mais uma unidade de processamento de inteiros e um novo canal de pipelining, o que aumenta substancialmente sua capacidade de processamento, já que pode agora completar três instruções em um ciclo de relógio.
- cache secundária L2 integrada na pastilha do processador — o maior aperfeiçoamento neste processador foi a inclusão de uma segunda memória cache, L2 no interior da pastilha, o que aumenta em muito seu desempenho na execução de instruções, com menos ainda acessos à MP.
- otimização para uso com código de 32 bits — melhora do desempenho especialmente em aplicativos e SO que já estejam com código de 32 bits.
- aumento da largura do barramento de endereços — enquanto desde os processadores 80386, passando pelos 486 e Pentium originais, o barramento de endereços tinha 32 bits de tamanho (espaço de endereçamento de 4G células de 1 byte cada), os processadores de 6.ª geração estenderam 4 bits a mais no barramento, passando a 36 bits e, por isso, um novo espaço de endereçamento de 64G células de 1 byte cada.
- aumento da capacidade de multiprocessamento — passando de 2 para até 4 processadores na placa-mãe, melhora principalmente no caso de seu emprego como servidor de rede.
- aperfeiçoamento na previsão de desvios — o buffer da unidade teve seu tamanho dobrado e sua precisão aumentada.

- execução especulativa de instruções — nesse caso, o processador executa uma opção das duas de desvio que sempre existem. Há 50% de chance de a opção estar certa (muitas vezes mais do que isso, como por exemplo, durante um looping) e, nesse caso, a execução já ocorreu quando se testa a condição. Caso a opção tenha sido a outra, o processador a executa sem perda de tempo (o que não se ganha é o tempo antecipado de execução como no caso anterior).
- término da execução de instruções fora de ordem — as três unidades de pipelining se mantêm sempre ocupadas, pois o processador procura o tempo todo uma instrução para executar, mesmo que esteja mais adiante na seqüência. Posteriormente, o resultado entrará em ordem.

O Pentium Pro é uma máquina híbrida, pois possui um núcleo de execução RISC (mais rápido) mantendo o resto de seus elementos dentro da arquitetura CISC. Para poder continuar a executar programas cujo código executável é de instruções CISC anteriores, ele possui um decodificador CISC antes do núcleo RISC.

A sua grande vantagem, a cache L2 integrada, revelou-se também um inconveniente. Primeiro, porque sendo integrada ao processador não permite expansão, a não ser trocando de processador também. Segundo,

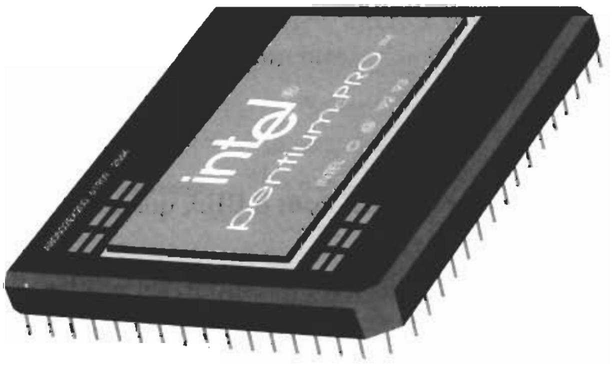


Figura 6.60(a) Microprocessador Pentium Pro.

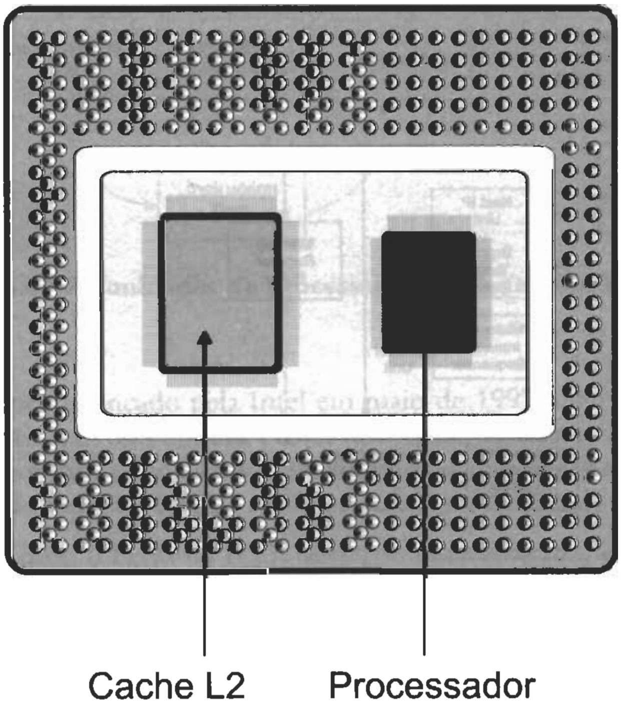


Figura 6.60(b) Pentium Pro.

pelo alto custo de fabricação, visto que em várias oportunidades pode ocorrer um erro na fabricação dos elementos de armazenamento e, nesse caso, o lote inteiro é perdido, com o conseqüente custo atribuído.

Conforme já mencionado, os processadores são capazes de, na média, decodificar, encaminhar para execução e executar três instruções em cada ciclo de relógio (clock), utilizando-se, para isso, de uma organização que possui 12 estágios de pipeline (um superpipeline), o qual suporta até a execução de instruções fora de ordem, de modo a não deixar nenhuma das unidades de execução ociosa.

Além disso, a arquitetura incorpora caches L1 e L2 no processador, sendo a L1 com 16KB (8KB de dados e 8KB de instruções) e a L2 com 256KB, ou 512KB ou até 1 Mbytes de memória SRAM, acoplada ao processador por um barramento de 64 bits operando na velocidade do relógio do processador.

A Fig. 6.61 mostra o diagrama em blocos do processador Pentium Pro, e a Tabela 6.1 descreve uma quantidade maior de dados sobre esses processadores. No diagrama apresentado, podem-se observar os diversos componentes (blocos) que constituem a estrutura organizacional deste processador (que serve também aos demais processadores da família P6) e os aperfeiçoamentos anteriormente mencionados. A Fig. 6.62 apresenta um diagrama resumido com os principais componentes do esquema de pipeline mencionado. São eles:

- fetch/decode unit,
- dispatch/execute unit,
- retire unit e
- instruction pool,

os quais funcionam com as duas partes da cache L1, com o BIU, que interliga a L1 com a L2 e o barramento do sistema.

A unidade de busca e decodificação (fetch/decode) é responsável por ler as instruções da cache L1 na sua seqüência normal (como vimos na primeira parte deste capítulo), decodificá-las e armazenar o microcódigo resultante no pool de instruções (instruction pool).

A unidade de programação e execução (dispatch/execute) é uma unidade de execução fora de ordem, responsável por programar a execução das microinstruções de acordo com a disponibilidade de dados e de recursos do processador, ou seja, uma instrução só pode ser executada, por exemplo, se a anterior foi completada.

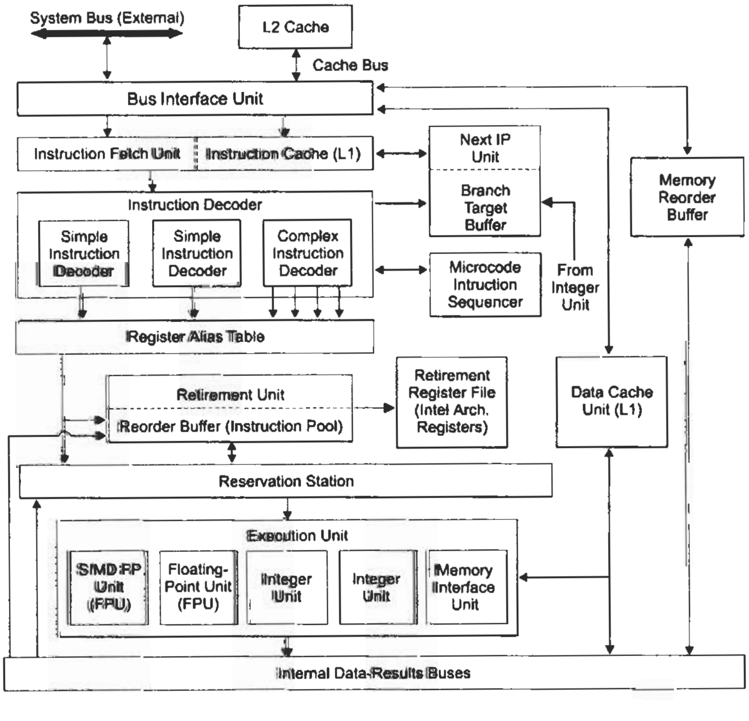

System Bus (External)
L2 Cache
Cache Bus
Bus Interface Unit
Instruction Fetch Unit
Instruction Cache (L1)
Next IP Unit
Branch Target Buffer
Memory Reorder Buffer
Instruction Decoder
Simple Instruction Decoder
Simple Instruction Decoder
Complex Instruction Decoder
Microcode Intruction Sequencer
From Integer Unit
Register Alias Table
Retirement Unit
Reorder Buffer (Instruction Pool)
Retirement Register File (Intel Arch. Registers)
Data Cache Unit (L1)
Reservation Station
Execution Unit
SIMD FP Unit (FPU)
Floating-Point Unit (FPU)
Integer Unit
Integer Unit
Memory Interface Unit
Internal Data-Results Buses

Figura 6.61 Diagrama em blocos dos processadores da família P6 (Pentium Pro e PII).

Neste caso, a unidade salta da segunda instrução para a seguinte (se esta puder), alterando, deste modo, a ordem de execução, visto que a terceira instrução será executada com a primeira, antes da segunda (esta menção, "primeira", "segunda", "terceira", etc. é apenas para efeito de facilitar a explicação). Os resultados parciais são armazenados nos 40 registradores internos, existentes no processador para esta finalidade.

A unidade de saída (retire unit) é responsável, finalmente, por colocar as instruções em ordem correta, verificando permanentemente os códigos de operação das instruções, armazenados no pool de instruções, para saber quais já terminaram a execução e não dependem de nenhuma outra anterior, que não tenha ainda sido executada. Ou seja, se esta unidade "retirou" a instrução n.º 5 (quinta instrução de uma dada seqüência), procura a de n.º 6 para poder manter a ordem final em seqüência.

Esta unidade pode "retirar" três instruções em cada ciclo de relógio; a "retirada" é realizada através da gravação (escrita) nos registradores existentes para este fim ou na memória. Tais registradores podem ser um dos 8 registradores de emprego geral ou um dos 8 registradores de ponto flutuante. Em seguida, a instrução é apagada do pool.

O membro seguinte, lançado pela Intel, logo substituiu o Pentium Pro, sendo denominado Pentium II, mostrado na Fig. 6.63, com seu novo tipo de encapsulamento, denominado SEC — Single Edge Contact, o qual se assemelha a um cartucho, que contém em seu interior o processador e a memória cache L2, bem como elementos para boa refrigeração dos componentes.

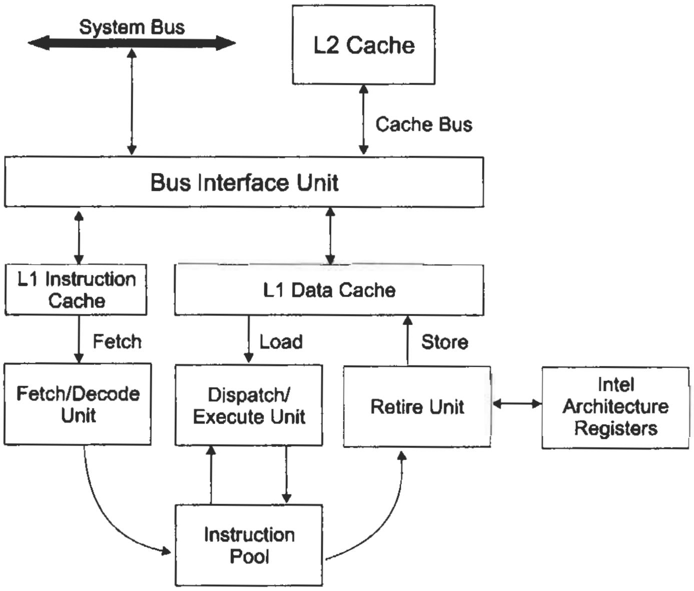


Figura 6.62 As unidades de processamento da família P6 da Intel.

O Pentium II foi oficialmente lançado pela Intel em maio de 1997, talvez com a principal finalidade de eliminar os problemas de fabricação do Pentium Pro devido à inserção da cache L2 na pastilha do processador. O Pentium II acrescentou, por isso, poucas inovações em relação ao processador anterior, exceto a retirada da L2, a criação do novo tipo de soquete, SEC, a ser inserido em novo tipo de soquete, denominado *slot 1*.

Além desses aperfeiçoamentos, o Pentium II foi fabricado em outras freqüências maiores, 233 MHz e 300 MHz, bem como teve sua cache interna, L1, aumentada de 16KB para 32KB, ainda dividida entre dados e instruções.

A terceira UCP da família P6, lançada pela Intel em janeiro de 1998 e denominada em código Deschutes, porém popularmente apenas Pentium II, podia operar na freqüência de 333 MHz. Tais processadores fabricados com tecnologia de 0,25 mícron de espessura possuem reduzido consumo de energia comparativamente

intel
pentium II
PROCESSOR

Figura 6.63 Processador Pentium II.

falando em relação ao Pentium II original (código Klamath), que é fabricado com tecnologia de 0,35 mícron de espessura. Posteriormente, esta mesma linha de processadores foi aperfeiçoada através do lançamento de um barramento de sistema operando a 100 MHz, o que permite operação do processador em velocidades da ordem de 350 MHz até 450 MHz. Com isso, também a velocidade das memórias cache L2 puderam ser aumentadas, acarretando tempos de acesso da ordem de até 4,4 ns.

Em seguida aos processadores Pentium II Deschutes, a Intel decidiu fabricar um Pentium menos potente, porém mais barato, de modo a fazer face à crescente concorrência da AMD e Cyrix. Este modelo de UCP foi denominado Celeron, o qual é em si um Pentium II, exceto que a cache L2 é retirada do sistema, barateando os custos de fabricação substancialmente, embora com redução de desempenho. Posteriormente, a Intel resolveu lançar um modelo de Celeron com 128KB de cache L2 (nome-código Mendocino), e aumentando progressivamente a velocidade de relógio do processador até 533 MHz, no final de 1999 e já lançando em 2000 modelos com 600 MHz e 700 MHz, produzidos com a nova tecnologia que permite espessuras de 0,18 mícron.

O Pentium III

O Pentium III foi lançado pela Intel em março de 1999, introduzindo como substancial aperfeiçoamento um conjunto de 70 instruções gráficas do tipo MMX, com o propósito de acelerar o processamento e, conseqüentemente, o desempenho de jogos em 3D, tanto quanto a tecnologia 3DNow! Da AMD (ver item seguinte sobre processadores não Intel). A Fig. 6.64 mostra o invólucro do Pentium III ainda com encapsulamento do tipo SEC.

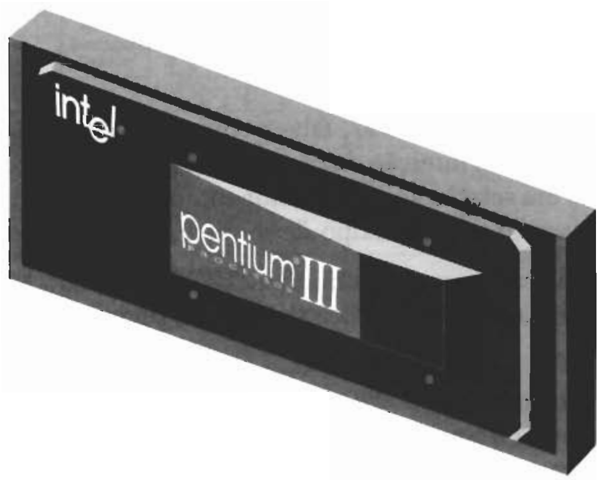


Figura 6.64 Processador Pentium III.

Os primeiros processadores Pentium III operavam na velocidade de 500 MHz, sendo bastante similares ao Pentium II exceto pela inclusão do já mencionado conjunto de instruções MMX e de novos registradores de ponto flutuante de 128 bits, justamente para emprego com aquelas instruções.

Em essência, as modificações do Pentium III em relação aos Pentium II podem ser resumidas no seguinte:

- *inclusão de 70 novas instruções, sendo 50 delas para permitir cálculos simultâneos de vários números* representados em ponto flutuante através da execução de uma única instrução, um fato extremamente útil em processamento gráfico, conforme já mencionamos anteriormente. As demais instruções são 12 instruções para a categoria de vídeo, as quais facilitam a codificação e a decodificação de dados de vídeo do tipo MPEG-2 à medida que vão sendo recebidos (*on the fly*) e 8 instruções que facilitam a interação entre a memória principal e a cache L2. Em conjunto essas 70 instruções são conhecidas como do tipo SSE (Streaming SIMD — ver Cap. 1 Extension) ou com o nome código de Katmai.
- Código de identificação único do chip (PSN — processor serial number), o qual tem gerado uma certa polêmica entre os que são a favor, por facilitar determinadas operações e marketing pela Internet, além de aperfeiçoar a segurança e criptografia para transmissões, e os que são contra, por considerar uma invasão de privacidade, além de contra-argumentar sobre segurança, pois alegam que fàcilmente hackers e crackers poderão identificar e acessar o tal número, de 96 bits, eletronicamente programado em cada pastilha.
- Novos registradores de 128 bits, para manipular 4 valores em ponto flutuante, acelerando o processamento de gráficos em 3D. No entanto, por se tratar de novos elementos inseridos na arquitetura do processador, não podem ser automaticamente reconhecidos pelo código atual dos sistemas operacionais. Certamente que as novas versões do Windows 98 e outros deverão conter os elementos apropriados para manipulação desses registradores.

6.6.5.2 Os Processadores de Outros Fabricantes

Os processadores da AMD e Cyrix

A AMD — Advanced Micro Devices é uma empresa fabricante de processadores, memórias e outros dispositivos semicondutores, conhecida atualmente no mercado como concorrente da Intel na fabricação de processadores. Foi fundada em maio de 1969, começando mesmo a ser conhecida quando, juntamente com a Cyrix, iniciou o lançamento de processadores clones dos produzidos pela Intel, os primeiros dos quais copiando os 80386. Já a Cyrix teve um caminho um pouco diferente, tendo sido incorporada pela National Semiconductors em 1987, e mais recentemente (julho 2000) foi vendida para a Via Technologies, grupo de Taiwan, fabricante de chips e semicondutores, também concorrente da Intel neste campo.

Como já mencionado, os primeiros processadores lançados pela AMD e Cyrix eram também chamados de 386, como os da Intel, sendo autênticas cópias dos mesmos. Tais processadores foram produzidos nas versões DX e SX, como a Intel, inclusive com freqüência de operação superior à da Intel (esta produziu 386 até 33 MHz de velocidade, enquanto a AMD e a Cyrix produziram processadores com 40 MHz, na versão DX apenas).

Já no caso dos processadores de quarta geração, da família 486, a AMD e a Cyrix puderam desenvolver e lançar no mercado produtos de sua própria lavra e não mais simples cópias dos da Intel.

A AMD produziu o chamado AMD 5x86, processador semelhante aos da Intel 486DX4, porém incrementou a multiplicação da freqüência do relógio interno, atingindo uma vez a mais e, por isso, sendo denominado 5x86. Embora utilizasse uma tecnologia de fabricação avançada, 0,35 mícron de espaçamento entre as "linhas" condutoras internas, ainda era um processador de quarta geração pelas suas características iguais às dos 486.

O 5x86 foi produzido para funcionar com 133 MHz de velocidade com um desempenho considerado muito bom para o tipo, sendo possível comparar-se seu desempenho ao de um Pentium 75 MHz, embora autores e analistas achem mais apropriado considerá-lo um 486DX4, com exceção da quantidade de memória cache L1, pois ele possui 16KB, enquanto os 486 vinham com 8KB.

Já a Cyrix produziu um processador da mesma faixa, também denominado 5x86 (ou M1), porém com características diferentes. O Cyrix 5x86 foi produzido de modo a operar com velocidades de 100 e 120 MHz, sendo este último considerado extremamente rápido para sua faixa, comparando-se seu desempenho ao de um Pentium 90 ou 100. Naturalmente, como o da AMD, este processador funciona sendo instalado nas placas-mãe fabricadas para processadores da família 486.

Com relação aos processadores de quinta geração, de características semelhantes aos Pentium, P5, da Intel, a AMD produziu o K5 e a Cyrix o processador 6x86.

Embora o AMD K-5 tenha sido projetado com características internas mais avançadas que o Pentium original (algumas delas são típicas do Pentium Pro, de sexta geração), ele continuou a ser considerado de quinta geração devido às baixas velocidades de relógio com que foi fabricado (75 MHz até 116 MHz). Entre suas principais características pode-se citar:

- arquitetura interna do tipo RISC, isto é, ele processa internamente em modo RISC, possuindo na entrada um decodificador x86 para que esse tipo de instrução possa ser convertido;
- 6 unidades pipeline — o processador possui 5 unidades de execução de inteiros e uma de ponto flutuante (o Pentium somente possui 2 unidades de inteiros);
- execução de instruções fora de ordem nas duas canalizações de inteiros;
- o pool de instruções é quatro vezes maior que o do Pentium, permitindo melhor desempenho na previsão de desvios;
- o K5 possui uma cache de instruções com 16KB (o dobro dos 8KB do Pentium), mas a cache de dados também tem 8KB, como nos Pentium.
- o mapeamento da memória cache no K5 é associativa por conjunto de 4, em vez do conjunto de 2 da cache dos Pentium.

O K5 não pode ser considerado um clone dos Pentium, pois é projetado com características próprias (não é uma cópia, como um clone se define) de funcionamento e desempenho, embora seja semelhante ao processador da Intel. A AMD fabrica seus próprios processadores, enquanto a Cyrix não era assim.

O processador desta geração lançado pela Cyrix foi denominado 6x86, embora não seja um processador de sexta geração. Da mesma forma que o K5 da AMD, o 6x86 não é um clone do Pentium, porém tem muitas semelhanças. Comparando-se os desempenhos, tem sido constatado que um 6x86 rodando a 133 MHz possui um desempenho igual ou superior ao de um Pentium rodando a 166 MHz.

Comparando-se o 6x86 ao Pentium original, o primeiro possui algumas vantagens sobre o Pentium:

- execução especulativa;
- aperfeiçoamento do mapeamento da cache L1, de conjunto de 2 para conjunto de 4;
- mais estágios de pipelining, passando de 5 para 7;
- unidade de previsão de desvios mais aperfeiçoada.

Um dos pontos mais interessantes da comparação entre os processadores é o fato do Cyrix 6x86 ser mais barato que o Pentium (também o AMD K-5 é mais barato que o Pentium).

No entanto, além das vantagens já mencionadas e de seu baixo custo, o Cyrix 6x86 também possui algumas desvantagens em relação ao Pentium, entre as quais:

- problemas de compatibilidade com algumas placas-mãe;
- não suporta multiprocessamento (os AMD também não);
- baixo desempenho no processamento em ponto flutuante em relação ao Pentium;
- problemas na execução de alguns programas, devido ao uso de certas instruções específicas do Pentium, não existentes no Cyrix.

Ambos os processadores, K5 e 6x86, não tiveram grande aceitação, acarretando problemas para seus fabricantes.

Com relação aos processadores de sexta geração, de características semelhantes aos Pentium, P6, da Intel, a AMD produziu o K-6, modelos 2 e 3 e a Cyrix, o processador 6x86MX (M2).

O processador AMD K-6 foi lançado em abril de 1997, graças à absorção pela AMD da empresa NexGen. Este processador foi considerado um sucesso dadas suas características avançadas e baixo custo comparativamente ao seu competidor, da Intel, superando os problemas da AMD com o K5. A Fig. 6.65 mostra alguns dos processadores da família K6.

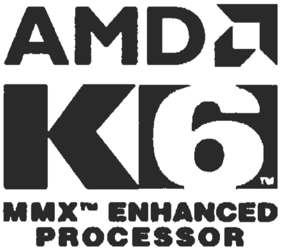


Figura 6.65(a) O processador AMD K6.

Figura 6.65(b) O processador AMD K6-2.

AMD
K6
2
3DNow!

Figura 6.65(c) O processador AMD K6-3DNow!

Dentre as principais características do K6, pode-se mencionar:

- tamanho elevado da memória cache L1 — 64KB, sendo 32KB para dados e o restante para instruções. Isso significa o quádruplo da cache L1 do Pentium Pro e o dobro do Pentium II;
- mais decodificadores de instrução — o K6 possui 4 dispositivos decodificadores, um a mais que o Pentium Pro e o Pentium II;
- mais unidades de execução de operações com inteiros — o K6 possui 6 dessas unidades, mais do que os processadores produzidos pela Intel ou Cyrix; com isso, permite um maior paralelismo na execução dos programas, com o conseqüente aumento de velocidade.

O K6 foi produzido inicialmente com velocidades de 166 MHz, 200 MHz e 233 MHz, tendo um desempenho muito bom, exceto no que se refere ao dispositivo de execução de operações em ponto flutuante.

Na mesma categoria de processadores, a Cyrix produziu o modelo 6x86MX (MII ou M2). Como os processadores da AMD, também esses se enquadram em um misto de 5.ª e 6.ª geração, sendo concorrentes do Pentium Pro e Pentium II em certas características e mais próximos do Pentium original em outras.

O processador M2 foi lançado em maio de 1997 com a informação de categoria semelhante ao Pentium MMX. Entre suas principais características, pode-se citar:

- processador superescalar, como os Pentium e AMD, porém com arquitetura puramente CISC, diferentemente daqueles citados, que optaram por um núcleo RISC (ver Cap. 11 para descrição da arquitetura RISC e sua comparação com a CISC);

- utilização do soquete tipo 7, como os demais processadores dessa geração;
- memória cache de 64KB;
- otimização de seu código interno para emprego em 32 bits;
- grande quantidade de registradores internos (32), de modo semelhante ao Pentium Pro (40) e ao AMD K-6 (48), que aceleram o tempo de processamento por evitar acessos à memória.

Prosseguindo em seus avanços tecnológicos, como aconteceu com a Intel, a AMD produziu em seguida o processador K6-2, depois o K6-3, para finalmente entrar na 7.ª geração com o processador K7, denominado Athlon. Enquanto isso, a Cyrix produziu o M3.

O AMD K6-2 foi lançado em maio de 1998, sendo produzido com tecnologia de 0,25 mícron de espaçamento entre suas trilhas internas e tendo cerca de 9,3 milhões de transistores em sua pastilha. Suas primeiras versões operavam em 300 MHz, sendo posteriormente fabricados processadores com até 450 MHz de freqüência. A Fig. 6.66 mostra o diagrama em blocos resumido desse processador, em que se pode observar o interface do barramento interno, de 100 MHz, e as duas memórias cache, de 32KB cada uma, para instruções e para dados.

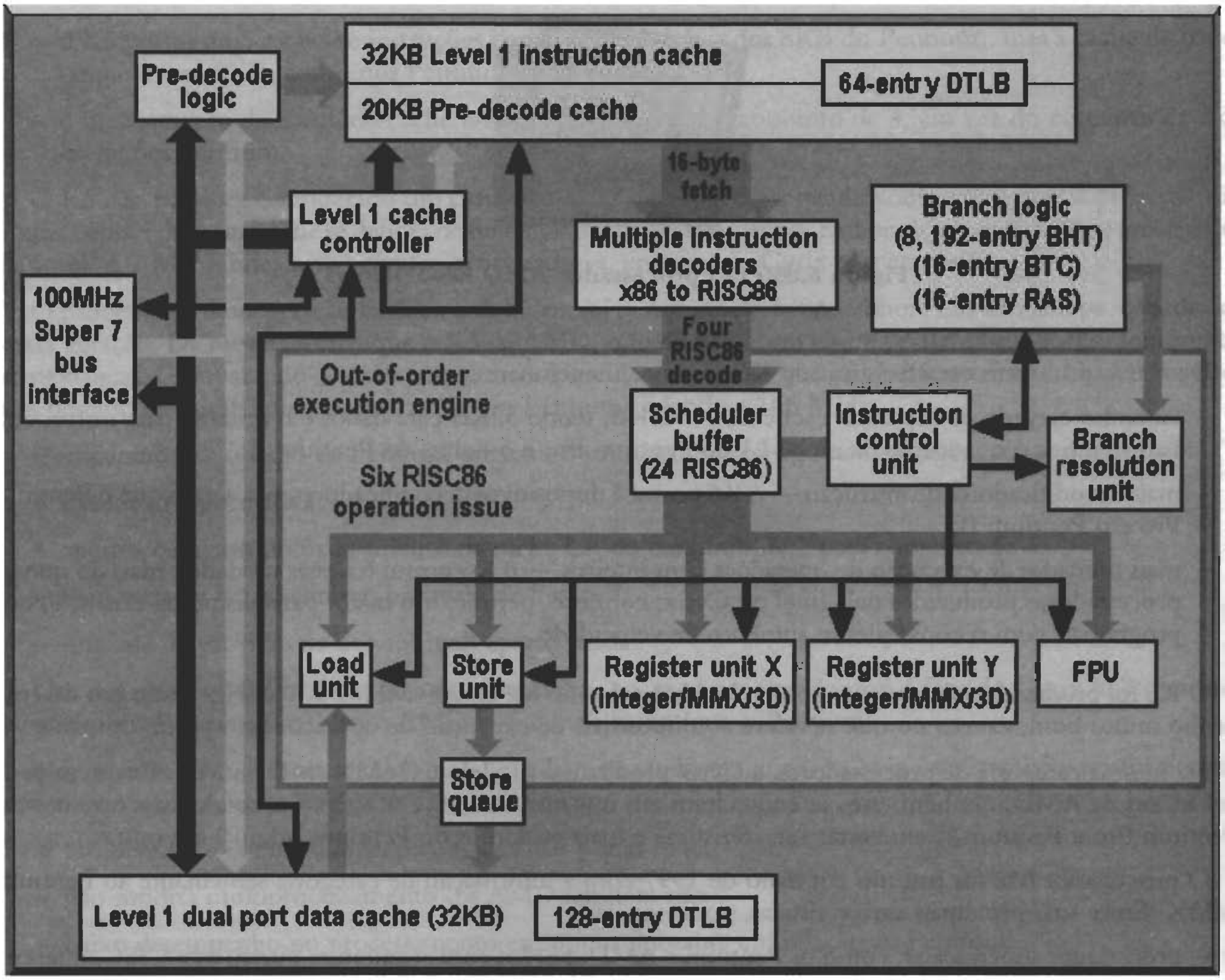

Pre-decode logic
32KB Level 1 instruction cache
64-entry DTLB
20KB Pre-decode cache
16-byte fetch
Level 1 cache controller
Multiple instruction decoders x86 to RISC86
Branch logic (8, 192-entry BHT) (16-entry BTC) (16-entry RAS)
100MHz Super 7 bus interface
Four RISC86 decode
Out-of-order execution engine
Scheduler buffer (24 RISC86)
Instruction control unit
Branch resolution unit
Six RISC86 operation issue
Load unit
Store unit
Register unit X (integer/MMX/3D)
Register unit Y (integer/MMX/3D)
FPU
Store queue
Level 1 dual port data cache (32KB)
128-entry DTLB

Figura 6.66 Diagrama em bloco do processador AMD K6-2.

Como os processadores Intel dessa geração, também o K6-2 incorpora em seu interior um núcleo de microarquitetura RISC. Além disso, o processador da AMD apresentou sensível aperfeiçoamento para a unidade de execução de instruções em formato de ponto flutuante.

Com o K6-2, a AMD também progrediu no mercado de computação gráfica e jogos, introduzindo uma tecnologia denominada 3DNow! (uma boa estratégia de marketing) (ver Fig. 6.65(c)), que consiste basicamente na incorporação de 21 instruções MMX ao conjunto de instruções do processador. Tais instruções efetuam processamento sofisticado em ponto flutuante, mais avançadas do que as primeiras instruções MMX lançadas pela Intel (ver item anterior). Naturalmente, para que o programa do jogo possa tirar partido do aumento de desempenho que essas instruções acarretam, é necessário que se possua o driver gráfico apropriado, que manipule diretamente com essas instruções. Modelos anteriores àquele lançamento não lucram nada com a inovação.

Em meados de 1999, a AMD anunciou o lançamento de uma nova versão do processador K6, denominada K6-3, nome-código de *Sharptooth*, a qual introduziu algumas características novas na arquitetura da família K6 e até no mercado como um todo, a saber:

- incorporação do nível L2 no interior da pastilha (como o Pentium Pro), porém, diferentemente do processador da Intel, que só possuía 2 níveis (L1 e L2) ambos internos, o K6-3 acrescentou um terceiro nível de cache, L3, externo ao processador.
- Utilização de um barramento local de 133 MHz, aumentando o desempenho das transferências de dados entre MP e UCP em relação ao barramento de 100 MHz.

O projeto de cache em 3 níveis torna melhor, sem dúvida, o desempenho do processador. No interior da pastilha do processador são inseridos os dois primeiros níveis, sendo L1 de 64KB, como nos processadores anteriores (32KB para dados e 32KB para instruções), e L2 com 256KB, esta funcionando também na velocidade igual à do processador, como nos Pentium Pro, Celeron e Xeon. Além dessas memórias, a AMD aproveitou-se do fato de estar utilizando uma placa-mãe que possui espaço para uma cache e criou o nível L3 de cache, com até 2MB. Um considerável aumento de desempenho.

Posteriormente ao K6-3, a AMD ainda lançou um modelo denominado K6-2+ e, finalmente, o AMD K-7, denominado Athlon, considerado o primeiro processador de 7.ª geração. A Fig. 6.67 mostra o processador inserido na placa-mãe, e a Fig. 6.68 apresenta seu diagrama em blocos resumido.

Como se pode observar na Fig. 6.67, o K7, Athlon, possui um formato semelhante ao Pentium III, porém ele é inserido em um soquete específico da AMD, denominado slot A, pois a Intel não licenciou seu soquete 1. Embora semelhantes, os soquetes são eletronicamente incompatíveis.

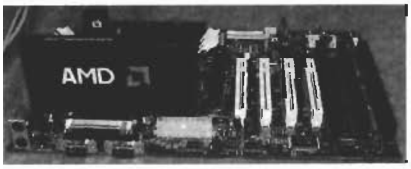


Figura 6.67 O processador AMD K7, inserido no soquete A da placa-mãe.

As principais características do K7, um processador de 22 milhões de transistores, são:

- Múltiplos decodificadores de instruções — 3 decodificadores para traduzir instruções x86 (CISC) em instruções tipo RISC, podendo se ter até 9 instruções executadas simultaneamente. O aumento de desempenho devido a esta facilidade é acentuado.
- Memória cache L1 com 128KB, operando na velocidade do relógio e memória cache L2, também interna na pastilha, com até 8MB (um acréscimo considerável). Até onde se pode verificar, no entanto, a AMD estará utilizando apenas 512KB dessa cache, cujo tamanho deverá crescer nos próximos lançamentos.

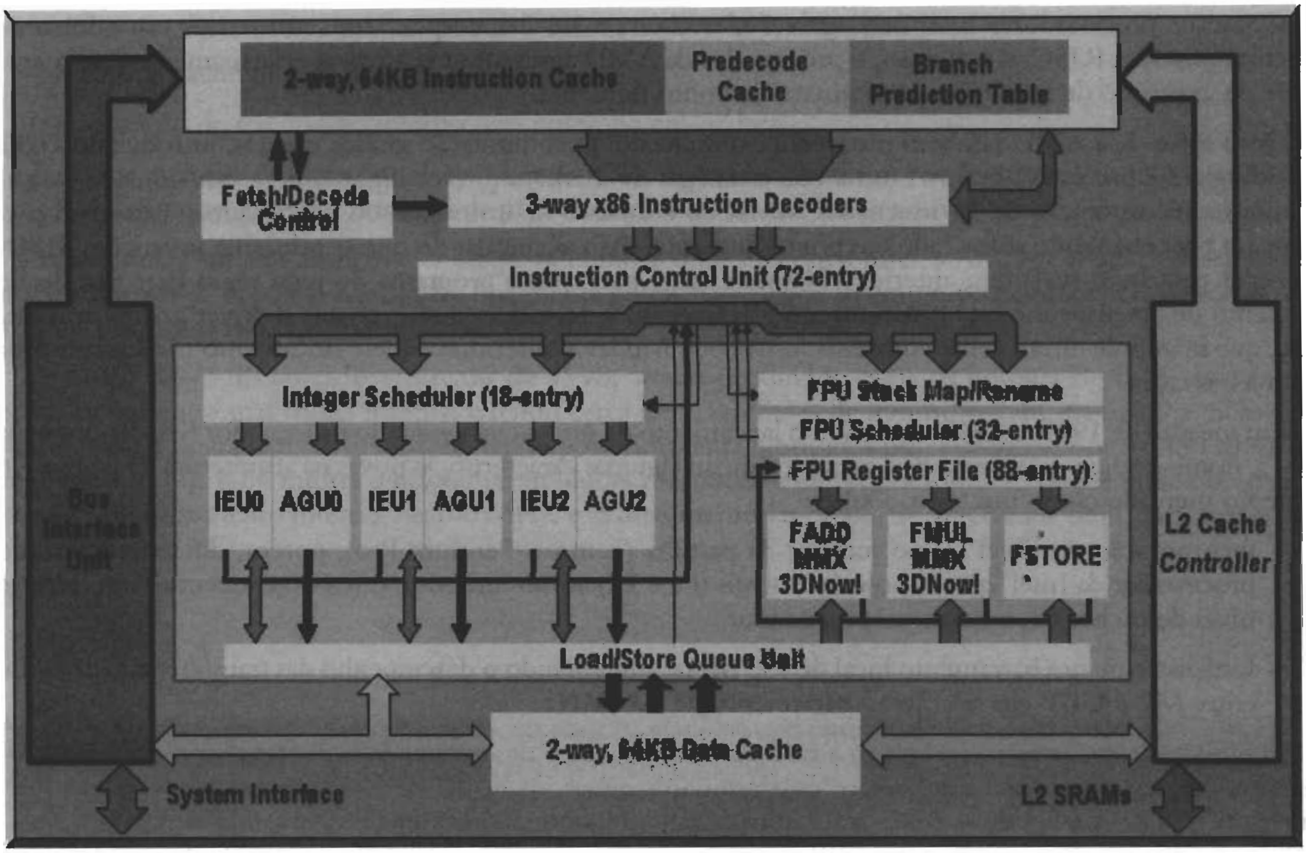

2-way, 64KB Instruction Cache
Predecode Cache
Branch Prediction Table
Fetch/Decode Control
3-way x86 Instruction Decoders
Instruction Control Unit (72-entry)
Integer Scheduler (18-entry)
IEU0
AGU0
IEU1
AGU1
IEU2
AGU2
FPU Stack Map/Rename
FPU Scheduler (32-entry)
FPU Register File (88-entry)
FADD MMX 3DNow!
FMUL MMX 3DNow!
FSTORE
Load/Store Queue Unit
2-way, 64KB Data Cache
Bus Interface Unit
L2 Cache Controller
System Interface
L2 SRAMs

Figura 6.68 Diagrama em bloco do processador AMD K7-Athlon.

- Velocidade inicial de 500 MHz.
- Capacidade de realizar multiprocessamento, o que acontece pela primeira vez nos processadores da AMD, capacitando o emprego desses processadores no mercado de servidores, até então dominado pela Intel e outros processadores mais potentes, como o Alpha da Compaq (antiga DEC). Esta possibilidade decorre principalmente devido ao emprego no K7 da arquitetura do processador Alpha (protocolo EV6), o qual implementa a topologia do tipo ponto a ponto. Ou seja, se há vários processadores no sistema, cada um deles obtém uma conexão ponto a ponto com a pastilha de apoio (chipset).
- Novo projeto na área de processamento de números representados em ponto flutuante, com o propósito de melhorar seu desempenho, uma deficiência crônica nos modelos anteriores de processadores AMD. Para tanto, foram incluídas 3 unidades de processamento de ponto flutuante, as quais podem executar instruções em paralelo (se uma não depender da outra, é claro) ou mesmo fora de ordem.
- Novo barramento de sistema de 200 MHz.

6.6.6 Elementos Auxiliares

6.6.6.1 Circuitos de Apoio (Chipsets)

O funcionamento de um sistema de computação envolve o controle do funcionamento de diversos componentes, bem como sua sincronização e cadência com que os eventos internos são realizados (dependente da freqüência do relógio utilizado). Entre as diversas atividades que devem ser controladas podem-se citar: as interrupções (ver Cap. 10), a transferência de dados entre a MP e a memória cache e entre esta e os registradores internos ao processador, o funcionamento da própria memória cache e da MP (decodificação de endereços, transferência de dados), o controle do funcionamento de um dispositivo denominado DMA — Direct

Memory Access, para transferência direta de dados entre os discos e a MP, conforme veremos no Cap. 10 e outros.

Nos primeiros sistemas de computação para microprocessadores (aqueles baseados nos processadores Intel 8080, 8085, Motorola 6800 e mesmo nos primeiros microcomputadores do tipo PC, que empregavam o processador 8088), as funções de controle mencionadas eram realizadas por circuitos separados, existentes em pastilhas específicas para cada finalidade. Se observarmos a placa-mãe desses antigos computadores poderemos verificar a existência de muitas dessas pastilhas instaladas.

Com o passar do tempo esta concepção de pastilhas individuais para o controle de cada função foi evoluindo para a integração das funções em uma ou poucas pastilhas (chips). A denominação inglesa *chipset* é, então, decorrente dessa integração. Em inglês, *chipset* significa conjunto (set) de chips, pois em uma pastilha (chip) se integram várias funções anteriormente realizadas por pastilhas (chips) individuais.

Atualmente, é possível encontrar placas-mãe com um ou dois ou até mesmo 4 chipsets, visto que, nem sempre, é possível integrar todas as funções requeridas pela placa-mãe em uma única pastilha (chip). A Fig. 6.69 mostra exemplos de chipsets, sendo que, como se pode observar, a Intel costuma denominar os chipsets que fabrica conforme o barramento de E/S que ele controla, seja PCI (e, nesse caso, ela o chama de PCIsets) seja AGP (denominado AGPsets).

Esta política foi primariamente estabelecida devido à necessidade permanente de redução de custos e de aumentar a compatibilidade entre componentes manufaturados por diferentes fabricantes.

intel
PCIset

Figura 6.69(a) Exemplo de um chipset.

Figura 6.69(b) Exemplo de um chipset.

Naturalmente que se torna mais barata a fabricação de uma pastilha para realizar várias tarefas em vez de várias pastilhas para realizar as mesmas tarefas, somente que de forma individual. Além disso, o projeto e o próprio uso se tornam mais simples no caso dos chipsets, o que permite, também, uma maior compatibilização no uso das placas-mãe.

Atualmente, há várias empresas que se destacam no mercado como fabricantes de chipsets, enquanto outras se especializam na fabricação de placas-mãe, uma situação normal, considerando o grau de padronização hoje existente, para o que o surgimento dos chipsets veio contribuir também.

Devido à natureza dos elementos de desempenho do sistema que os chipsets controlam, seu funcionamento tem um peso considerável no referido desempenho, devendo a escolha do modelo específico ser um requisito essencial.

De uma maneira geral, os chipsets controlam os seguintes elementos em um sistema de computação:

- o processador — que inclui a velocidade e o tipo do processador, e se a placa-mãe poderá suportar multiprocessamento;
- a memória cache — o tipo e o nível, bem como o funcionamento da cache e sua ligação com a MP e o processador;
- o funcionamento da MP — decodificação dos endereços e controle da transferência de dados;

- a sincronização dos eventos e o controle do fluxo dos bits;
- controle do funcionamento do barramento de E/S;
- gerenciamento da alimentação elétrica para o sistema.

6.6.6.2 Encapsulamento dos Circuitos em uma Pastilha (Chip) e Soquetes para Inserção de Processadores nas Placas-mãe

Encapsulamento dos circuitos em uma pastilha

A miniaturização dos elementos físicos (transistores, resistores, etc.) integrados em uma pastilha (chip) tem evoluído ao longo do tempo, tornando esses elementos cada vez menores. Para que se obtenha menores tempos de transferência de sinais entre os elementos, os pesquisadores têm conseguido construir pastilhas com espaçamentos internos cada vez menores. Esses espaçamentos entre os elementos, conhecidos como tamanho dos circuitos internos (ou trilhas), são medidos em mícrons, unidade de medida do sistema métrico, que é equivalente a 1 milésimo do milímetro. Atualmente, a tecnologia de construção das pastilhas (chips) de processadores permite tamanho de circuitos ou espaçamentos de 0,35 mícron, 0,25 mícron e, mais recentemente, 0,18, utilizado nos últimos processadores lançados (o espaçamento ainda menor, de 0,08 mícron, é o próximo passo, que tornará os processadores ainda mais capazes e velozes).

Uma pastilha de um processador como o Pentium ou K7, constituída exclusivamente com os elementos eletrônicos e condutores, é extremamente pequena, muito menor realmente do que o que vemos instalado na placa-mãe, como as Figs. 6.59, 6.60 e outras, mostradas neste livro. O que se vê é a pastilha encapsulada em um invólucro que permite uma melhor dissipação de calor e, principalmente, que permite inserir os pinos que serão soldados ou inseridos em um soquete. Dessa forma, a pastilha apresentada (externa) é muito maior do que realmente ela é (pastilha interna).

Ao longo do tempo surgiram diferentes processos de encapsulamento externo das pequenas pastilhas, sejam elas de um processador, assunto deste capítulo, ou de uma memória ou outro componente, construído com tecnologia de integração de circuitos.

O primeiro modelo de encapsulamento surgido foi o DIP — Dual Inline Package (ver Fig. 5.12), caracterizado por uma pastilha retangular, conectável nas placas de circuito impresso por uma dupla fileira de pinos, dispostos ao longo dos lados maiores do retângulo, como se pode observar na figura. Esse método de empacotar a pastilha interna, DIP, foi utilizado na fabricação dos processadores Intel 8080, 8088 e outros, e também na fabricação de alguns tipos de memória.

O pacote DIP deixou de ser utilizado para processadores na medida em que se aumentou a quantidade de sinais trocados entre o processador e o mundo exterior. Enquanto o 8080 possuía apenas 40 pinos, 20 de cada lado da pastilha, atualmente o Pentium Pro possui 387 pinos, o que tornaria uma pastilha DIP com comprimento imensamente longo devido à quantidade de pinos que teria.

Outro método criado para empacotar as pastilhas de processadores, que permite a utilização de maior quantidade de pinos sem crescer demasiadamente o tamanho do pacote, é denominado PGA — Pin Grid Array (ver figura dos processadores mais antigos, Figs. 6.1(a) e (d), por exemplo) e tem sido empregado em diversos processadores, desde o Intel 80286 até os últimos modelos de 5.ª geração (Pentium original, AMD K-5, etc.).

Neste método de construção da pastilha externa, esta é de formato quadrado ou retangular (ver Fig. 6.59) e seus pinos podem ser distribuídos pelos quatro lados do perímetro da pastilha. Para permitir uma inserção segura e padronizada, são construídos soquetes especiais na placa-mãe, específicos para inserção da pastilha. A descrição desses soquetes será apresentada mais adiante, ainda neste item.

As pastilhas do tipo PGA são construídas usando dois tipos básicos de material: cerâmica (e a pastilha é denominada CPGA), tendo sido a mais usada, e plástico (e a pastilha é denominada PPGA), tipo mais recentemente escolhido pelos fabricantes. O tipo de plástico (PPGA) é mais barato e termicamente superior ao tipo de cerâmica (CPGA), devido ao processo usado para dissipação de calor.

O método mais recente e eficaz de empacotar os chips internos é denominado SEC — Single Edge Contact, sendo uma mudança acentuada no formato do pacote em relação aos modelos anteriores de processadores. Isto foi especialmente necessário devido ao crescimento da quantidade de elementos internos dos processadores (muitos milhões de transistores) acarretado pela inclusão da cache de nível secundário (L2) na pastilha, caso do Pentium Pro. No entanto, o método SEC surgiu com o Pentium II, mesmo este tendo a cache L2 não mais inserida no mesmo pacote, porém a Intel decidiu colocar ambos os chips no mesmo conjunto e daí surgiu o método SEC, mostrado nas Figs. 6.63 e 6.70.

Na realidade, a tecnologia SEC consiste na criação de uma placa de circuito impresso em que a pastilha do processador (construída de forma semelhante ao método PGA) é inserida e também a pastilha da cache L2. A placa é, então, inserida em um soquete especialmente construído na placa-mãe. Este método permite uma grande velocidade de transferência entre o processador e a cache L2, visto que ela não está inserida na placa-mãe.

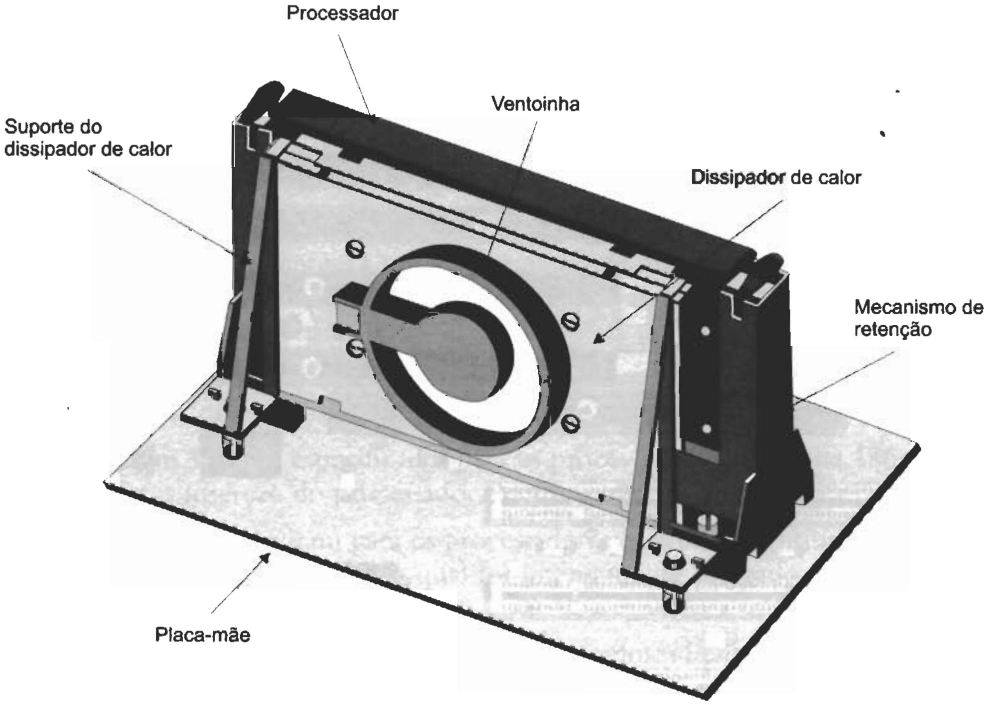


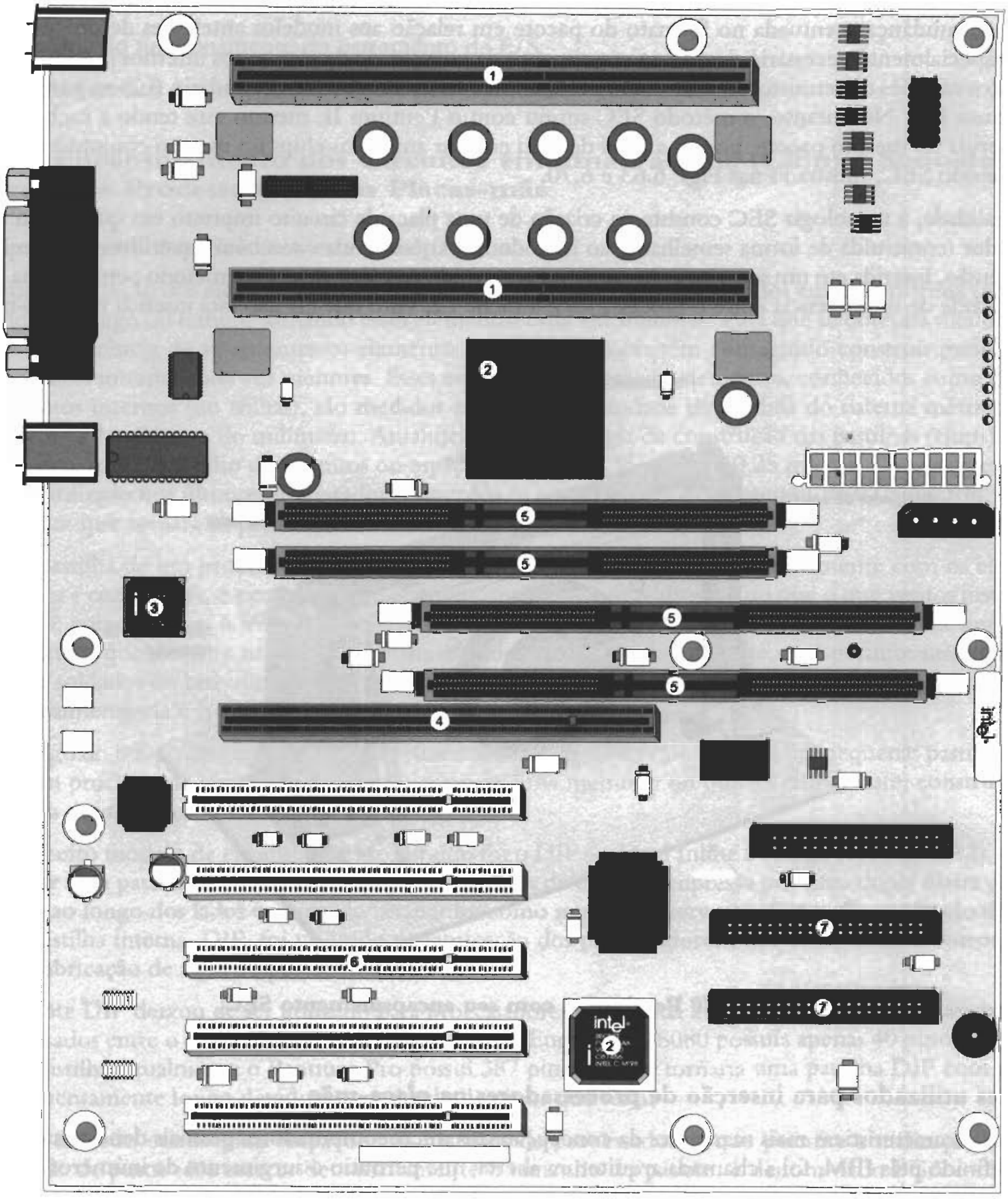
Figura 6.70 Pentium II com seu encapsulamento SEC.

Soquetes utilizados para inserção de processadores na placa-mãe

Uma das características mais marcantes da concepção dos microcomputadores pessoais denominados PCs, como definido pela IBM, foi a chamada arquitetura aberta, que permitiu o surgimento de inúmeros fabricantes dos componentes que constituem o sistema de computação em questão, diferentemente de uma arquitetura fechada, como dos computadores fabricados pela Apple, cujos elementos são patenteados, requerendo licença para fabricá-los.

Assim, há fabricantes específicos de diversos componentes do computador, entre os quais temos os que constroem a placa principal do sistema, denominada placa-mãe (*motherboard*), sobre a qual se inserem os demais componentes, entre os quais o processador. A Fig. 6.71 mostra um exemplo de placa-mãe, onde se podem observar os vários diferentes componentes, como soquete para o processador, para os módulos de memória e slots para placas de interface com os dispositivos de entrada/saída.

Nos primeiros anos de existência dos PCs, os usuários tendiam a adquirir o computador por inteiro e, deste modo, apesar de a arquitetura ser aberta, este conceito servia apenas para os fabricantes. Se a pessoa desejava atualizar o modelo do equipamento, substituía-o por completo e não apenas o seu motor (o processador), por exemplo.



Descrição dos itens numerados:

1. Soquetes dos processadores até 2 (slot para inserção de pastilhas tipo SEC).
2. Pastilhas do chipset.
3. Pastilha de controle da rede local.
4. Conector para inserção de placa de vídeo.
5. Conector para inserção de módulos de memória (nesta placa-mãe pode-se inserir até 2GB).
6. Conector para inserção de placas de dispositivos periféricos tipo PCI (até 5).
7. Conector para inserção de controlador de disco.

Figura 6.71 Exemplo de placa-mãe com seus principais elementos indicados por números.

Com o passar do tempo, a Intel passou a estimular os usuários a atualizar apenas o processador, se desejável, como também isto passou a servir de motivo para que as atualizações pudessem ser parciais (a placa de vídeo, ou os discos, e assim por diante).

Este conceito surgiu com a possibilidade de atualizar o processador, o que a Intel denominou overdrive. Para isso, a Intel necessitou estabelecer as características mecânicas e elétricas requeridas para o soquete onde se deveria inserir o processador, de modo que os fabricantes de placas-mãe pudessem construí-las apropriadamente. Surgiram, então, os padrões de soquete de processadores, que permitiram a fabricação de placas-mãe que iriam servir para uso dos processadores do momento, mas também de outros a serem concebidos mais adiante, desde que estes fossem projetados com a pinagem e demais características próprias para aquele soquete. A placa-mãe, assim, não teria que ser substituída com o surgimento de novos modelos de processadores, uma grande inovação e facilidade para o mercado, principalmente dos concorrentes da Intel, como a AMD e Cyrix, que se valeram da padronização dos soquetes para projetarem seus processadores de modo adequado, usando as mesmas placas-mãe.

O primeiro soquete, denominado soquete 1 (Socket 1) criado com esta finalidade, surgiu muitos anos depois do lançamento do primeiro PC e serviu especificamente aos processadores 486, dentro do conceito de overdrive. Ele tinha 169 pinos e se inseria nas placas-mãe dos 486.

Não faz parte do escopo deste livro descrever, em detalhes, as características de cada tipo de soquete lançado desde o soquete 1, porém, a título de informação para o leitor, podem-se mencionar as seguintes observações a respeito:

a) A Intel especificou até o momento 11 tipos de soquete, de números 1 a 8, soquete 370 e, em seguida, dois mais recentes, denominados slot 1 e slot 2, e não mais soquetes, que serviram para inserção do pacote modelo SEC (ver item anterior).

b) A AMD criou seu específico slot A para inserção dos processadores K7, Athlon, por incompatibilidade deste com as características elétricas do slot 1, conforme já mencionado anteriormente.

c) Os soquetes de 1 atém 3 foram especificados para os processadores da família 486, sendo que o soquete 3 permitia, também, a inserção do processador Pentium tipo overdrive.

d) Os soquetes 4 em dimante serviram para os processadores da família Pentium, sendo o mais conhecido atualmente o soquete 7, de 321 pinos, o qual foi intensamente utilizado pela AMD e Cyrix para seus processadores K6 e 6x86, respectivamente.

e) O soquete 8 foi projetado especificamente para os processadores Pentium Pro e posteriormente Pentium II. É o único que suporta o Pentium Pro.

f) O soquete 370 foi projetado especificamente para o processador Intel Celeron.

g) Os slot 1 e 2 trouxeram uma acentuada mudança nas características desses elementos, tendo sido projetados para atender ao formato e tipo do módulo SEC.

Na Tabela 6.1 está indicado o soquete característico de cada tipo de processador mencionado.

6.6.6.3 Overclocking

Conforme já foi descrito anteriormente neste capítulo, um sistema de computação opera cadenciado pelas "batidas de um pêndulo", ou seja, pelos pulsos emitidos pelo componente que chamamos de relógio. Cada microoperação é realizada pelo estímulo de um desses pulsos. Quanto menor for o intervalo entre um pulso e outro (ou quanto maior for a freqüência de emissão dos pulsos) mais rapidamente as tais microoperações serão realizadas e, naturalmente, o sistema funcionará mais rápido como um todo.

Os processadores são projetados para funcionar em certa faixa de velocidade, isto é, podem aceitar diferentes intervalos entre os pulsos, dentro de uma certa quantidade, acima ou abaixo da qual eles deixam de funcionar adequada e confiavelmente.

Definindo de uma maneira objetiva e simples, *overclocking* (não se usa uma tradução apropriada para esta palavra, mencionando-se, de um modo geral, a própria palavra em inglês) significa a possibilidade de o usuá-

Tabela 6.1 Características de Processadores

Processador	Modelo	Fabricante	Ano de Lançamento	Velocidade Barramento MP-MHz	Velocidade Processador MHz	Tecnologia de Fabric.	Intervalo Interno (mícrons)	Quant. Transist. no Processador (milhões)	Largura do BD (bits)	Largura do BE (bits)	Capac. Máxima da MP (bytes)
8080		Intel							8	16	64K
8086	xxxxx	Intel	1978	4,77-10	4,77-10	NMOS, CMOS	3,0	0,029	16	20	1M
8088	xxxxx	Intel	1979	4,77	4,77	NMOS, CMOS	3,0	0,029	8	20	1M
80286	xxxxx	Intel	1982	6-20	6-20	CMOS	1,5	0,134	16	24	16M
80386	DX	Intel	1985	16-40	16-40	CMOS	1,5	0,275	32	32	4G
80486	DX	Intel	1989	25-50	25-50	CMOS	1,0	1,2	32	32	4G
5x86	xxxxx	AMD	1995	33	133	CMOS	0,35	xxxxx	32	32	4G
5x86M1	xxxxx	Cyrix	1996	33-50	100-120	CMOS	0,65	2,0	32	32	4G
Pentium	P5	Intel	1993	60-66	60-200	Bipolar CMOS	0,8-0,35	3,1	64	32	4G
Pentium	MMX	Intel	1997	66	166-233	CMOS	0,35	4,5	64	32	4G
6x86	M1	Cyrix	1995	50-75	100-150	CMOS	0,06-0,5	3,0	64	32	4G
K5	xxxxx	AMD	1995	50-66	75-116	CMOS	0,35	4,3	64	32	4G
Pentium Pro	P6	Intel	1995	60-66	150-200	Bipolar CMOS	0,6-0,35	21,0	64	36	64G
Pentium	II	Intel	1997	66	233-333	CMOS	0,35-0,25	7,5	64	36	64G
K6	xxxxx	AMD	1997	66	166-266	CMOS	0,35-0,25	8,8	64	32	4G
6x86MX	M2	Cyrix	1997	60-75	150-187	CMOS	0,35	6,0	64	32	4G
Pentium	III	Intel	1999	100-133	500-900		0,25-0,18				
K7	Athlon	AMD	1999								

Tabela 6.1 Características de Processadores (continuação)

Processador	Modelo	Tipo do Soquete	Tipo do Encapsulamento	Pinos do Soquete	Cache L1	Tipo da Cache L1	Cache L2 (na placa-mãe)	Processadores de Inteiros	Processador Ponto Flutuante	Estágios Pipeline	Tamanho dos Registradores (bits)
8080								1	xxxxx		8
8086	xxxxx	DIP	DIP	40	Não	xxxxx	Não	1	8087	1	16
8088	xxxxx	DIP	DIP	40	Não	xxxxx	Não	1	8087	1	16
80286	xxxxx	PGA	PGA	68	Não	xxxxx	Não	1	80287	1	16
80386	DX	PGA	PGA	132	Não	xxxxx	Não	1	80387	4	32
80486	DX	Soq 1, 2 e 3	PGA	168	8KB	Unificada	Até 256KB	1	1 - no chip	5	32
5x86	xxxxx	Soquete 3	PGA	168	16KB	Unificada	Até 256KB	1	1 - no chip	5	32
5x86M1	xxxxx	Soquete 3	PGA	168	16KB	Unificada	256 KB	1	1 - no chip	5	32
Pentium	P5	Soq 4, 5 e 7	PGA	273 e 296	16KB	D-8 e I-8	256 a 512K	2	1 - no chip	5	32
Pentium	MMX	Soquete 7	PGA	296	32KB	D-16 e I-16	256 a 512K	2	1 - no chip	6	32
6x86	M1	Soquete 7	SPGA	296	16KB	Unificada	256 a 512K	2	1 - no chip	7	32
K5	xxxxx	Soq 5 e 7	PGA	296	24KB	D-8 e I-16	256 a 512K	5	1 - no chip	5	32
Pentium Pro	P6	Soquete 8	SPGA	387	16KB	D-8 e I-8	256K a 1M	4	1 - no chip	14	32
Pentium	II	Slot 1	SEC	242	32KB	D-16 e I-16	512K	5	1 - no chip	14	32
K6	xxxxx	Soquete 7	SPGA	296	64KB	D-32 e I-32	256K a 1M	6	1 - no chip	6	32
6x86MX	M2	Soquete 7	SPGA	296	64KB	Unificada	256 a 512K	2	1 - no chip	7	32
Pentium	III	Slot 1 e 2					512K a 2M				
K7	Athlon	Slot A									

rio fazer o processador operar em uma freqüência de relógio (mais pulso por segundo) maior do que sua especificação (ou seja, aumentar as "batidas do pêndulo" mais do que o fabricante considera tolerável).

É claro que, procedendo assim (realizando o overclocking), o processador funcionará mais rápido, porém este é um procedimento não recomendável, principalmente para usuários que não dominem amplamente os conhecimentos sobre o funcionamento dos processadores, em particular, e do sistema, como um todo, pois "envenenar" o motor (o processador) para acelerá-lo pode acarretar diversos tipos de problema no funcionamento do sistema, a começar pelo seu "travamento".

Isto tudo é possível devido a vários fatores, tais como: os componentes do sistema (especificamente o barramento que interliga MP e UCP ou, nos sistemas mais modernos, que interliga a MP e cache L2, bem como o que interliga as caches L1 e L2 e L1 e registradores) funcionam em velocidades diferentes, por razões de capacidade, flexibilidade e outras. Além disso, os fabricantes divulgam valores de velocidade de um processador (por exemplo, Pentium 600 MHz), que é bem abaixo do máximo que o processador pode funcionar; isto é uma segurança que o fabricante usa para garantir que, na velocidade especificada ele sempre funcionará com segurança, em outras maiores nem sempre. Esta última política, por si só, já induz a possibilidade de uso do overclocking.

Como mencionado antes, os componentes funcionam com velocidades diferentes, porém relacionadas por um fator multiplicador (este procedimento de uso do fator multiplicador facilita consideravelmente o projeto e funcionamento do sistema). Ou seja, o processador usualmente opera em uma velocidade de relógio múltipla da velocidade do barramento do sistema. Por exemplo, se o barramento opera em 66 MHz (velocidade muito comum), utilizando-se um fator multiplicador de 2.5, obtém-se uma velocidade do processador de 166 MHz. Este fator multiplicado pode ser ajustado por alteração na placa-mãe (jumper) ou no firmware (se a placa não tiver jumpers).

6.6.6.4 Organização de Dados na Memória do Tipo Big Endian e Little Endian

Poucos itens ou elementos podem ser considerados completamente padronizados pelo mercado da computação, de modo que diferentes sistemas possam se comunicar sem problemas ou sem necessidade de elementos intermediários de interface. Assim é com o conjunto de instruções e arquitetura de registradores, que a Intel e a Motorola, por exemplo, projetam de modo diferente, como também quanto ao conjunto de códigos de caracteres (temos ASCII, EBCDIC, Unicode, etc.).

No entanto, talvez grande parte das pessoas não possa imaginar que também a ordem com que os bytes e bits são armazenados em um sistema não é padronizada, o que acarreta, por isso, alguns problemas, requerendo elementos de intermediação quando se transferem dados entre dois sistemas projetados por fabricantes diferentes e que usem métodos diferentes de ordenar os dados internamente.

Quando mencionamos a ordem dos bytes queremos indicar que os bytes que constituem uma palavra podem ser armazenados na MP da direita para a esquerda ou vice-versa, da esquerda para a direita. Em outras palavras, supondo que um certo sistema utilize palavras de dados de tamanho igual a 32 bits (4 bytes ou 8 algarismos hexadecimais), um dado com o valor binário de: 01100001010100011100111010010100, pode ser representado em hexadecimal assim: 6151CE94.

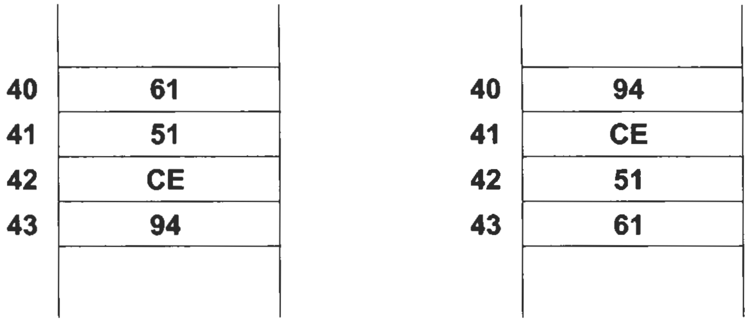
Este valor poderá ser armazenado na memória de um sistema de dois modos:

1) 6151CE94, ou da esquerda para a direita (são armazenados a partir da célula de menor endereço para a célula de maior endereço, conforme mostrado na Fig. 6.72(a).

2) 94CE5161, ou da direita para a esquerda (são armazenados a partir da célula de maior endereço para a célula de menor endereço, conforme mostrado na Fig. 6.72(b)).

O método mostrado inicialmente na Fig. 6.72(a) é denominado *big endian*, sendo empregados em máquinas fabricadas pela Motorola, IBM e Sun, e o outro método (Fig 6.72 (b)) é denominado *little endian*, adotado pela Intel, Compaq e outros.

Os termos *big endian* (maior valor-*big*-em primeiro lugar-menor endereço) e *little endian* (menor valor-*little*-em primeiro lugar) foram inseridos no jargão da computação por um artigo publicado em 1981 por D. Cohen, citando o problema e relacionando-o a um episódio mencionado no livro *As Viagens de Gulliver*, de Jonathan Swift, em que ele satiriza os políticos ingleses da época e descreve um povo encontrado por Gulliver (um personagem de ficção) que foi à guerra para decidir qual a melhor maneira de quebrar ovos, se pelo maior (big) lado ou se pelo menor (little) lado.



(a) Big endian (b) Little endian

Valor em hexadecimal: 6151CE94

Figura 6.72 Exemplo de ordenação de bytes na memória.

Embora haja, como indicado acima, fabricantes que adotam um ou outro método para armazenar os bytes de um valor multibyte, praticamente todos adotam um único método para armazenar os bits em um byte, o método big endian, ou seja, no caso, p. ex., do armazenamento do valor hexadecimal 61 (1 byte), teríamos:

0 1 1 0 0 0 0 1

Naturalmente, qualquer dos dois métodos teria pouco significado se tratarmos de um sistema individualmente. No entanto, se ligarmos duas máquinas em uma rede de comunicação de dados e uma das máquinas (a que usa o método big endian, por exemplo) transferir dados para a outra (que usa o método little endian), teremos um problema de entendimento entre elas sobre qual dado está sendo referido, devido à ordem diferente, o que implicará a necessidade de um interfaciamento para compatibilizar os dois métodos.

EXERCÍCIOS

1) Considere o diagrama de tempo para execução pipelining mostrado na Fig. 6.32. Refaça aquele diagrama considerando agora que são 4 os estágios que estão sendo executados.

2) Descreva as funções básicas de uma UCP, indicando os seus componentes principais.

3) Quais são as funções da Unidade Aritmética e Lógica — UAL?

4) O que é e para que serve o ACC?

5) Qual é o componente de um processador que determina o período de duração de cada uma de suas atividades e controla o sincronismo entre elas?

6) Quais são as funções da unidade de controle de um processador?

7) Seria possível realizar o projeto de um processador no qual o tamanho em bits do CI fosse diferente do tamanho do REM? Nesse caso, qual dos dois registradores deveria ter maior tamanho? Por quê?

8) Considere um computador cuja MP é organizada com N células de 1 byte cada uma. As instruções interpretadas pela UCP possuem três tamanhos diferentes: as do tipo A possuem 16 bits; as do tipo B

têm 32 bits e as do tipo C possuem 48 bits. Considerando que o código de operação de cada uma tem um tamanho fixo e igual a 8 bits e que os programas executados nesse processador são constituídos de uma mistura dos três tipos de instruções, imagine um processo prático para incremento automático do CI após a execução de cada instrução de um programa.

9) Considere um processador cujo ciclo de instrução não possua a etapa de incremento automático do valor do CI. Imagine um método alternativo que permita a execução do programa.

10) Qual é e onde se localiza o registrador cujo conteúdo controla a seqüência de processamento das instruções de um programa?

11) Considerando as instruções a seguir, indique a quantidade de ciclos de memória despendida para realizar seu ciclo de instrução completo (explicite a quantidade de ciclos de leitura e de escrita quando for o caso):

ADD Op.	ACC ← ACC + (Op.)
SUB Op.	(Op.) ← ACC − (Op.)
ADD Op.1, Op.2	(Op.l) ← (Op.l) + (Op.2)
INCR	ACC ← ACC + 1
LDA Op.	ACC ← (Op.)

12) Qual é o registrador cujo conteúdo determina a capacidade de memória de um computador? Justifique.

13) Considere um computador com 64K células de memória, instruções de um operando, tendo possibilidade de ter um conjunto de 256 instruções de máquina. Considerando que cada instrução tem o tamanho de uma célula, que é o mesmo tamanho da palavra do sistema, qual o tamanho, em bits, do ACC, CI e RDM? Qual é o total de bits dessa memória?

14) Um computador tem um REM de 16 bits e uma barra de dados de 20 bits. Possui instruções de um operando, todas do tamanho de uma célula de memória e de mesmo tamanho da palavra. Ele foi adquirido com apenas uma placa de 4K de memória.

Pergunta-se:

a) Qual o tamanho, em bits, do RDM e CI?

b) Seria possível aumentar-se a capacidade de armazenamento dessa memória? Até quanto? Por quê?

c) Qual a quantidade máxima de instruções de máquina que poderia existir nesse computador?

15) Um computador possui um conjunto de 128 instruções de um operando. Supondo que sua memória tenha capacidade de armazenar 512 palavras e que cada instrução tem o tamanho de uma palavra e da célula de memória, pergunta-se:

a) Qual o tamanho em bits do REM, RDM, RI, ACC e CI?

b) Qual a capacidade da memória, em bytes?

c) Se quisesse alterar o tamanho das instruções para 17 bits, mantendo inalterado o tamanho do REM, quantas novas instruções poderiam ser criadas?

16) Quando se fala que um determinado microcomputador A é um micro de 8 bits e que um outro micro B é de 16 bits, a que estamos nos referindo? Ao tamanho da célula de MP ou ao tamanho da palavra? Qual a base desses dois conceitos (palavra e célula)?

17) Considere um computador que possua uma UCP com CI de 16 bits e RI de 38 bits. Suas instruções possuem dois operandos do mesmo tamanho (16 bits), além, é claro, de um código de operação.

Pergunta-se:

a) Qual o tamanho da instrução?

b) Qual o tamanho do campo do código de operação?

c) Considerando que a configuração básica dessa máquina é de 16 Kbytes de memória, até que tamanho pode a memória ser expandida?

18) A figura a seguir apresenta uma memória de 256 células em que cada célula (ou palavra) contém 16 bits. Na figura, cada retângulo simboliza uma célula de memória; o número hexadecimal que está dentro do retângulo representa o seu conteúdo, e o número colocado ao lado de cada um indica o endereço da célula (retângulo).

00	0010
01	A0FD
02	0000
A4	1123
A5	C1305
A6	B200
FD	4040
FE	21F8
FF	09A5

Pergunta-se:

a) Qual a capacidade total da memória, em bits?

b) Supondo que, no início de um ciclo de instrução, o conteúdo do CI (contador de instrução) seja o hexadecimal A5 e que cada instrução ocupe uma única célula (palavra), que instrução será executada?

c) Supondo que o conteúdo do REM (registrador de endereços de memória) tenha o valor hexadecimal FD e que um sinal de leitura seja enviado da UCP para a memória, qual deverá ser o conteúdo do RDM (registrador de dados de memória) ao final do ciclo de leitura?

19) Escreva a seqüência de microoperações que devem ser realizadas para completar-se o ciclo das seguintes instruções:
a) SUB Op. ACC ←ACC + (Op.)
b) JMP Op. CI ← Op.
c) INC Op. (Op.) ← (Op.) + 1

20) Qual a razão por que não se deve utilizar vários somadores parciais para somar números com vários algarismos?

21) Utilizando o modelo de somador paralelo da Fig. 6.30, descreva o processo de soma entre os números A e B, sendo: A = 1100 e B = 0111.

22) Faça o mesmo para A = 1100 e B = 1110.

23) Qual é a diferença entre uma microinstrução horizontal e uma vertical?

24) Utilizando o diagrama de tempo da Fig. 6.36, descreva, passo a passo, uma operação de escrita em um barramento síncrono.

25) Em um processador que funcione com a técnica *pipelining*, qual é o problema de desempenho que pode surgir em relação a instruções de desvio condicional? Indique uma possível solução para este problema.

Índice

A

- Ábaco, 9
- Acerto (hit), 144
- Adaptador, 377
- AGP, 215
- Álgebra Booleana, 87
- Algoritmo, 5, 24
- Análise léxica, 355
- AND, 264
- Armazenamento, unidade de, 122
- Arquitetura, 2, 175
  - RISC, 222
  - superescalar, 234
  - x86, 226
- Arquivo, 31
- ASCII, 263, 382
- Assembly, 199

B

- Barramento, 126, 372
  - assíncrono, 217
  - ciclo do, 215
  - de controle, 212, 126
  - de dados, 126, 181, 234
  - de endereços, 126
  - de expansão, 213, 373
  - do sistema, 213, 373
  - largura de um, 214
  - local, 213, 373
  - síncrono, 219
- Base, 40, 442
  - conversão de, 445
- BCD (Binary Coded Decimal), 48, 307-308
- Big endian, 254
- Binária, 49
  - divisão, 52
  - multiplicação, 51
  - soma, 49
  - subtração, 49
- BIOS (Basic Input Output System), 386
- Bit, 29, 110
- BIU (Bus Interface Unit), 232
- Bloco, 400
- BSC, 382
- Byte, 29

C

- Cache
  - de RAM, 146
  - L1, 146
  - L2, 147
- Caractere, 29, 262
- CD-ROM, 424
- Célula, 110
- CENTRONICS, 383
- Chip, 97
- Chipsets, 246
- Circuito integrado - CI, 97
- Circuito somador, 204
- Circuitos combinatórios, 93
- CISC - Complex Instruction Set Computers, 175, 188, 429
- Código, 29
  - de operação, 189, 323, 352
  - de representação, 460
  - ligação, 258
  - objeto, 258
  - representação de caracteres, 29
- Código-fonte, 359
- Código-objeto, 351
- Compilação, 353, 360
- Compilador, 261, 274, 431
- Complemento, 282, 285
  - à base, 282
  - à base menos um, 293-294
  - a 2, 287
- Computador digital, 64

D

- Dados, 258, 261
- Decodificador, 100
- DIP (Dual Inline Package), 248
- Disco(s)
  - flexível, 405
  - magnético, 402
  - óticos, 424
  - rígido, 402
  - tempo de latência, 403
  - tempo de SEEK, 403
  - tempo de transferência, 403
- Display, 387
- Disquete
  - fator de bloco, 406
  - floppy-disk, 405
- DMA - Acesso Direto à Memória, 413
- DMA, 216

E

- EBCDIC, 263
- Endereçamento, 190
  - modo de, 190, 323, 330, 334
    - base mais deslocamento, 343
    - direto, 332
    - imediato, 331
    - indexado, 338
    - indireto, 333
  - por registrador, 335
- Endereço, 111, 122
- ENIAC, 12, 13
- Entrada, 26
- Entrada/saída (E/S), 373-374, 384
  - Acesso Direto à Memória (DMA - Direct Memory Access), 408
  - interrupção, 408
  - métodos, 408
  - por programa, 408
- Escrita (write), 125
- Expressão lógica, 80

F

- Falta (miss), 144
- Fita magnética, 399
- Flip-flop, 102, 156
- FPU (Floating Point Unit), 203
- Freqüência
  - horizontal, 389
  - vertical, 389

G

- Gap semântico, 429
- Gravação (write), 109

H

- HDLC/SDLC, 382

I

- IAS, 14
- IEEE-754, 315
- IESA, 215
- Impressora(s)
  - a laser, 395, 397
  - coloridas, 421
  - de cera aquecida, 395
  - de impacto, 395
  - de jato de tinta, 395, 397
  - matriciais, 395
  - por sublimação de tinta, 395, 423
  - sem impacto, 395
  - transferência térmica de cera e, 432
- Instrução(ões), 26, 116
  - assembly, 200
  - ciclo, 170, 183, 191
    - ADD Op, 197
  - com dois operandos, 327
  - com três operandos, 326
  - com um operando, 329
  - conjunto de, 322
  - contador, 186
  - de desvio, 365
  - de máquina, 26, 170, 187, 323
  - de um operando, 351
  - decodificador, 186
  - registrador, 186
  - tamanho das, 325
- Interface, 377

Interpretação, 359, 361
Interrupção, tratamento de, 413
ISA, 215

L

LCD, 420
Leitura (read), 109, 125
Leitura, ciclo de, 216
Ligação ou linkedição, 357-358
Linguagem(ens), 5
  ADA, 262
  de alto nível, 8, 349
  de máquina, 5, 347
  de montagem (Assembly Language), 348, 351
  de programação, 5, 347
Little-endian, 179, 254
Localidade, 143
  espacial, 143
  temporal, 143

M

Máquina de diferenças, 10
Memória(s), 26, 108, 113-115, 158
  Burst Extended Data Out (BEDO) DRAM, 158
  cache, 117, 144
  capacidade, 131
  ciclo, 114
  de acesso aleatório (ou randômico), 136
  Direct Rambus DRAM (DRDRAM), 158
  DRAM (Dynamic RAM), 137, 156
  EEPROM (Electrically ou Electronically EPROM), 140
  EPROM (Erasable PROM), 140
  Extended Data Out (EDO) DRAM, 158
  Fast Page Mode (FPM) DRAM, 158
  Flash ou Flash-ROM, 140
  meio magnético, 115
  meio ótico, 115
  principal, 119, 122
  PROM (Programmable Read Only Memory), 140
  RAM, 136, 158
  ROM (Read Only Memory), 137, 138
  secundária, 120
  SRAM (Static RAM), 137, 155
  Synchronous DRAM (SDRAM), 158
  volátil, 114
MFLOPS (milhões de operações de ponto flutuante por segundo), 18, 36, 433
Microcomputadores, 20
Microoperações, 222
Microprograma, 219
MIMD (Multiple Instruction stream, Multiple Data stream), 3
MIPS (milhões de instruções por segundo), 18, 432
MISD (Multiple Instruction stream, Single Data stream), 3
Montador, 201
Mouse, 407
  track-ball, 408

N

Notação
  científica, 297
  posicional, 39
Números fracionários, 272

O

8514A, 394
Operadores lógicos, 264, 266, 269
Operando, 323
Organização de um computador, 2
Overclocking, 251
Overflow, 207, 260, 273, 281, 296

P

Palavra, 30, 122, 177
  de dados, 377
Pastilha, 97
PC, 177
PCI, 215
Pentium, 235
Periféricos, 372
PGA (Pin Grid Array), 248
Pipeline, 172, 208
Pipelining, 181, 431, 434
Pixel, 392
Ponto flutuante, 300, 303
Porta(s), 67, 96
  AND, 67
  lógica (gate), 65
  NAND, 72
  NOR, 75
  NOT, 71
  OR, 69
  Wired-And, 96
  Wired-Or, 96
  XOR, 78
POWER (Performance Optimization With Enhanced RISC), 436
Power PC, 428
Processador central, 168
Processamento
  de dados, 1, 172
  eletrônico de dados, 1
Programa, 4, 31, 347
  de computador, 25

R

RAM (Random Access Memory), 118, 123
Recarregamento (refresh), 157
Registrador(es), 116, 126, 175
  de dados, 175
  de Dados da Memória (RDM), 126, 187
  de Endereços da Memória (REM), 126, 177
  instruções, 116
  PSW (Program Status Word), 177
Registro, 31
  físico, 400
Relógio, 182, 183
  freqüência, 184
Representação, 274
  de dados, 260
  decimal, 307
  normalizada, 299
  ponto fixo, 274
  ponto flutuante, 297
RISC - Reduced Instruction Set Computers, 172, 175, 188, 428
ROM (Read Only Memory), 124
RS/6000, 428, 436

S

Scanners, 423
SEAC (Standard Eastern Automatic Computer), 324
SEC (Single Edge Contact), 249
SIMD (Single Instruction stream, Multiple Data stream), 3, 172
Sinal e magnitude, 275-276
SISD (Single Instruction stream, Single Data stream), 3, 172, 184, 208
Sistema, 3
  de computação, 371
  de numeração, 440
  posicional, 440
  processamento de dados, 4
SPARC, 428, 435
SPECmark, 433
SRAM, 118
SVGA, 394

T

Tabela
  de símbolos, 353
  verdade, 65
Teclado, 384, 415
  debouncing, 416
  QWERTY, 417
Tempo de acesso, 108, 113, 116, 127
Temporariedade, 115
TFT, 420
Tipo numérico, 272
Transmissão
  assíncrona, 380
  paralela, 376, 383
  serial, 376, 379
  síncrona, 382

U

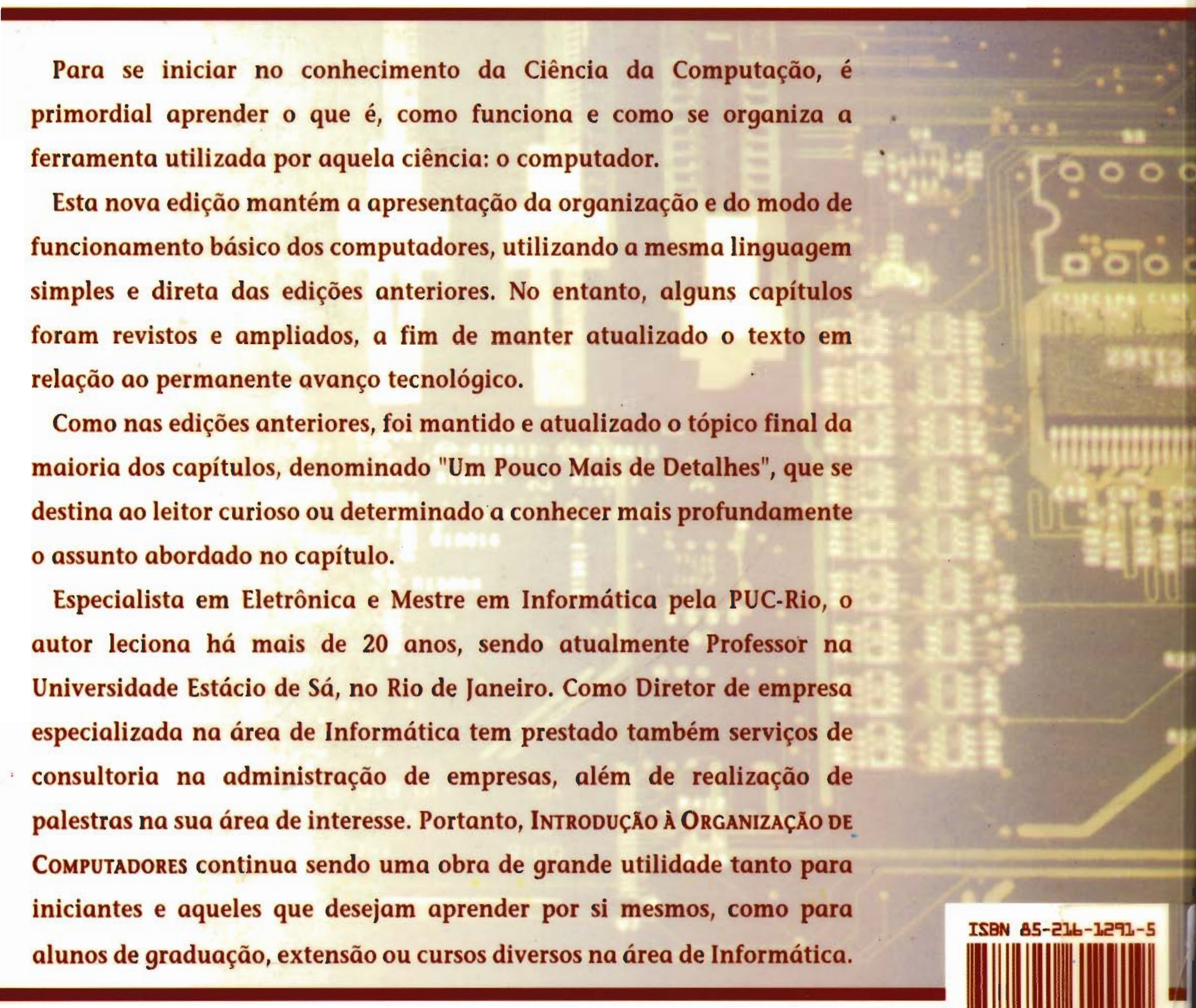
UAL, 174, 203
UC, 219
UNICODE, 263
Unidade
  Aritmética Lógica - UAL (ALU - Arithmetic and Logic Unit), 116
  Central de Processamento - UCP, 26
  de Controle (UC), 182
USART, 383
USB, 215, 383

V

Vazão, 37
VGA, 394
Vídeo, 386, 418
  colorido, 393
  de gás plasma, 421
  dot pitch, 394
  modo entrelaçado, 393
  não-entrelaçado, 393
  placa de, 390
  resolução de, 394
Volatilidade, 114
VRC, 387

INTRODUÇÃO À ORGANIZAÇÃO DE COMPUTADORES

Mário A. Monteiro



Para se iniciar no conhecimento da Ciência da Computação, é primordial aprender o que é, como funciona e como se organiza a ferramenta utilizada por aquela ciência: o computador.

Esta nova edição mantém a apresentação da organização e do modo de funcionamento básico dos computadores, utilizando a mesma linguagem simples e direta das edições anteriores. No entanto, alguns capítulos foram revistos e ampliados, a fim de manter atualizado o texto em relação ao permanente avanço tecnológico.

Como nas edições anteriores, foi mantido e atualizado o tópico final da maioria dos capítulos, denominado "Um Pouco Mais de Detalhes", que se destina ao leitor curioso ou determinado a conhecer mais profundamente o assunto abordado no capítulo.

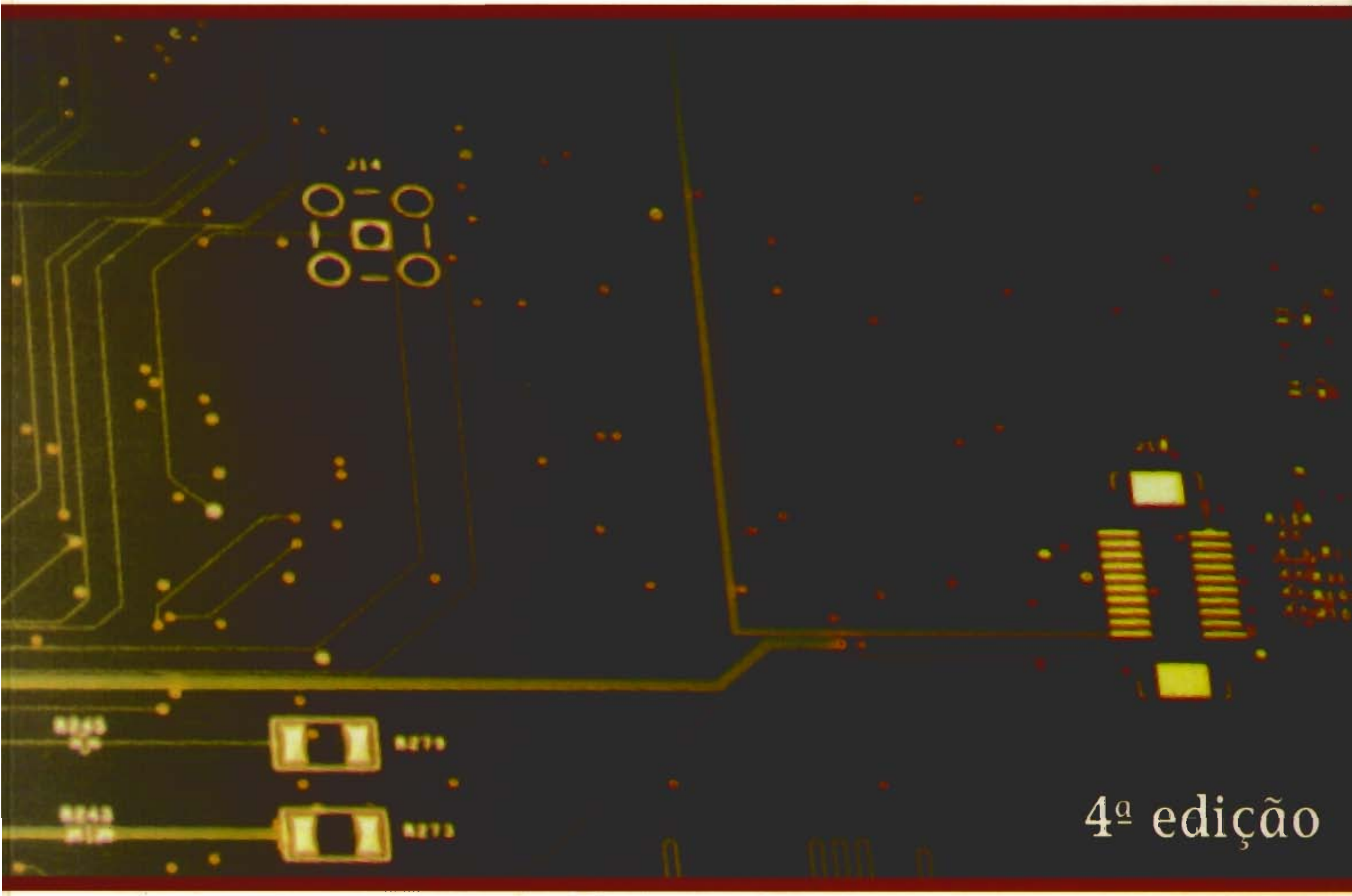
Especialista em Eletrônica e Mestre em Informática pela PUC-Rio, o autor leciona há mais de 20 anos, sendo atualmente Professor na Universidade Estácio de Sá, no Rio de Janeiro. Como Diretor de empresa especializada na área de Informática tem prestado também serviços de consultoria na administração de empresas, além de realização de palestras na sua área de interesse. Portanto, INTRODUÇÃO À ORGANIZAÇÃO DE COMPUTADORES continua sendo uma obra de grande utilidade tanto para iniciantes e aqueles que desejam aprender por si mesmos, como para alunos de graduação, extensão ou cursos diversos na área de Informática.

LTC LIVROS TÉCNICOS E CIENTÍFICOS EDITORA

ISBN 85-216-1291-5

9 788521 612919

INTRODUÇÃO À ORGANIZAÇÃO DE COMPUTADORES



4ª edição

Mário A. Monteiro

UNIBAN
Tombo 00229379

LTC

INTRODUÇÃO À ORGANIZAÇÃO DE COMPUTADORES

ABPDEA
Associação Brasileira para
a Proteção dos Direitos
Editoriais e Autorais
RESPEITE O AUTOR
NÃO FAÇA CÓPIA
www.abpdea.org.br

[redacted]

7

Representação de Dados

7.1 INTRODUÇÃO

Já sabemos que um computador funciona através da execução sistemática de instruções que o orientam a realizar algum tipo de operação sobre valores (numéricos, alfabéticos ou lógicos). Estes valores são genericamente conhecidos como *dados*.

Quer desejemos calcular uma expressão matemática complexa, quer o objetivo seja colocar uma relação de nomes em ordem alfabética, como também a tarefa de acessar a Internet e até a manipulação do mouse, tarefas que requerem ou não a execução de operações matemáticas, todas elas necessitam do emprego de instruções[1] que ativem operações com os dados. Estes dados podem ser valores numéricos (no cálculo de expressões matemáticas), valores alfabéticos (caracteres) ou ainda valores apenas binários (lógicos). De qualquer modo, tanto as instruções quanto os dados estão sempre armazenados internamente sob a forma de uma seqüência de 0s e 1s, os algarismos binários, que constituem a linguagem da máquina.

Quando digitamos os valores dos dados, estes são convertidos internamente em um código de armazenamento como se fossem caracteres alfabéticos (nesse instante, eles são tratados como um texto), conforme mostrado nos Caps. 6 e 10 (ver item 10.3.1), isto é, quando introduzimos, por exemplo, um dado cujo valor decimal é 143, este número é digitado algarismo por algarismo, é claro, primeiro o algarismo 1, depois o algarismo 4 e, em seguida, o algarismo 3. Logo, o sistema de computação segue o mesmo processo, recebendo estes algarismos não como o número 143, mas sim um texto com caracteres codificados segundo o código de armazenamento interno utilizado (na maioria dos casos é o código ASCII — ver item 7.3 e Apêndice B).

Nesse exemplo, seriam introduzidos (digitados) os seguintes valores binários:

00110001 (algarismo 1 em ASCII) 00110100 (algarismo 4) 00110011 (algarismo 3)

Qualquer que tenha sido a linguagem de programação utilizada para escrever o programa, este deverá ser convertido para um outro programa equivalente, porém em linguagem de máquina, denominado código-objeto e, em seguida, completado através do processo de ligação, tornando-se um código executável pelo processador (os processos de compilação, que geram código-objeto e código de ligação, que dão origem a códigos executáveis, são apresentados no Cap. 9). A referida conversão (compilação) também inclui os dados, que deverão ser alterados de modo a estarem em uma forma apropriada para utilização pela unidade aritmética e lógica (UAL). Em outras palavras, se o programa que vamos executar contiver um comando do tipo:

`X:= A + B;`

---

[1] Neste texto, para distinguir o assunto, denominaremos *instruções* quando se tratar de linguagem de máquina e *comandos* quando tratarmos de instruções em linguagens de alto nível. Há pessoas que denominam indistintamente instruções, acrescentando o atributo "de máquina" ou "linguagem de alto nível".

antes da sua execução teremos que, de alguma forma, introduzir um valor numérico correspondente a "A" e um outro valor para "B", de modo que esses valores sejam lidos pela UCP e somados na UAL. Para efetivar a soma, a UAL executa, passo a passo, uma série de microoperações (um algoritmo), como, por exemplo, verificar o sinal dos números, efetuando uma ou outra ação diferente, conforme o valor dos sinais, como será descrito mais adiante.

No entanto, dependendo da forma com que o dado foi definido no programa pelo programador (ao escrever o programa em uma linguagem de alto nível), o referido algoritmo poderá ser diferente (em cada passo serão realizadas microoperações diferentes, de acordo com o algoritmo realizado), embora o resultado final seja o mesmo. Ou seja, se o programador definiu o dado como um valor inteiro, ele será representado de uma forma diferente, por exemplo, da forma que ele seria internamente representado se o programador tivesse definido o mesmo dado como sendo um valor fracionário. E a operação aritmética, embora sendo a mesma (uma soma, por exemplo), teria sua efetivação estabelecida por algoritmos diferentes, um para o caso de soma com números representados sob a forma de inteiros e outro para o caso de uma soma com números representados sob a forma de fracionários. Por exemplo:

Se $A = +5$ e $B = -3$, então, após a execução do comando, teremos: $X = +2$.

Se, por exemplo, A e B forem representados internamente de uma forma binária simples e direta, com 16 bits (mais adiante veremos que se trata do caso de os valores estarem sendo representados como inteiros com sinal), teríamos para cada um:

$A = +5 = 0000000000000101$

$B = -3 = 1000000000000011$

Observação: Não vamos, por ora, explicar o que significa o bit 1 mais à esquerda da variável B.

E o algarismo para a UAL realizar a soma dos dois valores seria mais ou menos igual ao que mentalmente fazemos para a mesma operação.

No entanto, poderíamos usar uma máquina calculadora que utilizasse uma outra forma de representar os mesmos valores (+5 e −3), como, por exemplo, a notação científica matemática:

$A = +0{,}005 \times 10^{+3}$

$B = -0{,}003 \times 10^{+3}$

Observação: Para números inteiros e tão pequenos, esta forma de representação nunca é utilizada. Está servindo apenas para tornar a explicação mais clara.

Nesse caso, o algoritmo que a UAL deve executar certamente será diferente, embora, no final, os resultados sejam idênticos.

Esta diferença de formas de representação e respectivos algoritmos de realização das operações matemáticas é bastante útil, pois cada uma tem uma aplicação onde é mais vantajosa que a outra. Cabe ao programador a escolha da forma a ser utilizada pelo sistema, podendo ser explícita, quando ele define as variáveis e constantes em seu programa, ou implícita, ao deixar que o compilador faça sua própria escolha. A seguir são apresentados alguns exemplos de definição de variáveis em determinadas linguagens, indicando-se, em cada caso, o tipo de dados correspondente, estabelecido internamente no sistema.

Exemplo 7.1 Definição de variáveis em Pascal

Inteiros	int I
String sem limite de tamanho	
String com 30 caracteres	
Ponto flutuante	real V
Array de 5 valores inteiros (este não é um tipo primitivo de dados)	

Exemplo 7.2 Definição de variáveis em C

Inteiros	int I
String sem limite de tamanho	char *S
String com 30 caracteres	char S{30}
Ponto flutuante	double V
Array de 5 valores inteiros	int I[5]

(este não é um tipo primitivo de dados)

Exemplo 7.3 Definição de variáveis em Visual Basic

Inteiros	Dim I As Integer
String sem limite de tamanho	Dim S As String
String com 30 caracteres	Dim S As String*30
Ponto flutuante	Dim V As Double
Array de 5 valores inteiros	Dim I [4] As Integer

(este não é um tipo primitivo de dados)

Exemplo 7.4 Definição de variáveis em Delphi

Inteiros	var I: Integer
String sem limite de tamanho	var S: String
String com 30 caracteres	var S: String [30]
Ponto flutuante	var V: Double
Array de 5 valores inteiros	var I: Array[0...4] of Integer

(este não é um tipo primitivo de dados)

Este capítulo trata justamente das diversas formas de representação de dados ("tipos de dados", em linguagens de programação) utilizadas e compreendidas pelo hardware dos sistemas modernos. Não discorreremos especificamente sobre todos os possíveis tipos de dados aceitos pelas linguagens de programação, visto que grande parte deles consiste na especificação lógica de representação do dado e não na sua forma primitiva de uso pelo hardware. Trataremos apenas dos tipos primitivos, inteligíveis à UCP e às instruções de máquina.

Um dos aspectos mais importantes do processo de representação de dados em um computador e, conseqüentemente, do projeto propriamente dito do processador se refere à quantidade de algarismos que cada dado deve possuir na sua representação interna, o que afetará consideravelmente o tamanho e capacidade de inúmeros componentes do sistema. Por exemplo, o projetista da Intel ou da AMD ou da Motorola, ao definir os elementos de projeto de um novo processador, pode estabelecer que os números inteiros serão representados internamente com valores binários de 32 algarismos cada um (32 bits é o termo mais usual). Isso significará uma série de providências correlatas, como a definição do barramento interno de dados, da quantidade de fiação, do tamanho da Unidade Aritmética e Lógica (ver Cap. 6) para suportar operações aritméticas e lógicas com valores de 32 bits, e assim por diante.

Pode acontecer, por exemplo, que uma operação aritmética de multiplicação realizada com valores representados em 32 bits produza um resultado de valor maior que o maior número representável com 32 bits. Nesse caso, ocorre o que se denomina *estouro* da representação (*overflow*), o que está descrito com maiores detalhes mais adiante, neste capítulo.

Assim, diferentemente de nossos cálculos usando papel e lápis, cujo único limite é o tamanho do papel (embora possamos emendar várias folhas e torná-la enorme), em computação precisa-se ter atenção aos limites impostos pela quantidade máxima de bits dos valores representados e dos diversos componentes da máquina (registradores, barramento etc). Tais limites afetam a precisão dos resultados, como veremos mais adiante.

7.2 TIPOS DE DADOS

Conforme podemos observar pelos exemplos mostrados anteriormente, quando um programador elabora um programa ele precisa definir para o sistema como cada dado deverá ser manipulado, isto é, ele deverá (explícita ou implicitamente) determinar o tipo de cada dado declarado. Assim, por exemplo, a declaração

VAR ANOS: INTEGER;

indica para um programa compilador Pascal (ver Cap. 9 sobre compiladores) que o número que foi armazenado na memória no endereço correspondente à variável ANOS deverá ser um valor inteiro, a ser convertido para uma forma binária com 16 bits. Porém, se a declaração fosse do tipo:

VAR ANOS: REAL;

o valor numérico a ser armazenado em REAL teria uma forma diferente, representada em notação científica.

As palavras INTEGER e REAL foram, pois, interpretadas de modo diferente, acarretando alterações significativas, tanto no modo de organizar os bits que representam um número, quanto na seqüência de etapas do algoritmo de execução de uma operação aritmética com o número.

Talvez algum leitor possa, ainda neste instante (antes de ter lido o capítulo por completo), se perguntar que formas diferentes são essas e, principalmente, que algoritmos diferentes são esses para realizar a mesma operação aritmética com os mesmos valores. Se, por exemplo, tivéssemos os valores (decimais):

1249 e 3158

e desejássemos somar ambos os valores, utilizando duas formas diferentes para sua representação. Nesse caso, os passos (algoritmo) para realização da referida operação de soma seriam diferentes conforme a forma de representação utilizada. Senão vejamos:

Exemplo 7.5 Soma de 1249 e 3158, representados com sua forma natural, de valores inteiros

A efetivação da soma se realiza pela soma parcial de cada algarismo individualmente especificado (mais o algarismo correspondente ao "vai 1"), como fazemos na vida cotidiana, usando papel e lápis. No presente exemplo serão quatro somas, uma para cada parcela de algarismo.

$$
\begin{array}{r}
0\,1\,1 \\
1\,2\,4\,9 \\
+\,3\,1\,5\,8 \\
\hline
4\,4\,0\,7
\end{array}
$$

Exemplo 7.6 Soma de 1249 e 3158, representados na forma matemática denominada notação científica

A efetivação da soma com os valores representados em notação científica se realiza de acordo com as seguintes etapas:

1) Converter cada valor para a representação em notação científica:

   $1249 = 0{,}1249 \times 10^{+4}$ e $3158 = 0{,}3158 \times 10^{+4}$

2) Em seguida se realiza a soma propriamente dita. Neste caso, como os expoentes são iguais (ambos de valor igual a +4) e a parte fracionária está com as vírgulas alinhadas, ambos com 4 algarismos depois da vírgula, a soma se realiza somente através da adição, algarismo por algarismo, dos valores fracionários, 0,1249 e 0,3158.

$$
\begin{array}{r}
0\,1\,1 \\
0{,}1\,2\,4\,9 \\
+\,0{,}3\,1\,5\,8 \\
\hline
0{,}4\,4\,0\,7
\end{array}
$$

3) O resultado final será 0,4407 × 10^{+4}.

Caso os expoentes não fossem iguais, seria necessário antes que eles fossem igualados, através de modificação (por multiplicação ou divisão uma ou mais vezes por 10 da parte fracionária de um dos números) do valor da parte fracionária de um dos números.

Ainda neste capítulo, no item referente à representação e às operações matemáticas em ponto flutuante, o leitor poderá encontrar uma explicação mais detalhada do assunto.

De um modo geral, as seguintes formas de dados são mais utilizadas nos programas atuais de computador (formas primitivas, entendidas pelo hardware):

- dados sob forma de caracteres (tipo caractere);
- dados sob forma lógica (tipo lógico);
- dados sob forma numérica (tipo numérico).

Outras formas mais complexas são permitidas em certas linguagens modernas (como tipo REGISTRO, tipo ARRAY, tipo INDEX, tipo POINTER etc.). No entanto, durante o processo de compilação, os dados acabam sendo convertidos finalmente nas formas primitivas já mencionadas, para que o hardware possa entendê-las e executá-las.

A Fig. 7.1 apresenta um quadro contendo a nomenclatura das formas primitivas mais conhecidas e dos tipos de dados correspondentes.

Cada linguagem implementa um ou mais desses tipos primitivos de dados. A linguagem ADA, por exemplo, utiliza INTEGER (para valores inteiros), FLOAT (para valores fracionários), CHARACTER (para símbolos alfanuméricos) e BOOLEAN (para variáveis do tipo lógico). A linguagem Pascal denomina INTEGER, REAL, CHAR e BOOLEAN, respectivamente. E as demais linguagens utilizam nomes semelhantes, conforme podemos observar nos exemplos já apresentados.

FORMA		**TIPOS**
Caractere		**Caractere**
Lógico		**Lógico**
Numérico	Ponto fixo	Sinal e magnitude Complemento a 1 Complemento a 2
	Ponto flutuante	
	Decimal	

Figura 7.1 Tipos primitivos de dados.

7.3 TIPO CARACTERE

A representação interna de informação em um computador é realizada através da especificação de uma correspondência entre o símbolo da informação e um grupo de algarismos binários (bits). Isso porque o computador, possuindo somente dois símbolos (0 ou 1) para representação, requer mais de um bit para identificar todos os possíveis símbolos que constituem as informações usadas pelo homem e que precisam ser armazenadas e processadas na máquina.

Como poderemos representar, com apenas dois símbolos (0 e 1), todos os caracteres alfabéticos (maiúsculos e minúsculos), algarismos decimais, sinais de pontuação e de operações matemáticas etc., necessários à elaboração de um programa de computador?

Isso é obtido através de um método chamado codificação, pelo qual cada símbolo da nossa linguagem tem um correspondente grupo de bits que identifica univocamente o referido símbolo (caractere). A codificação é, então, a forma de representar caracteres (alfabéticos ou numéricos) armazenados no computador.

Desde o advento da computação, vários códigos de caracteres foram desenvolvidos para representação interna de informações nessas máquinas. Dentre eles podemos citar:

BCD — Binary Code Decimal — grupo de 6 bits/caractere, permitindo a codificação de 64 caracteres (praticamente não é mais empregado).

EBCDIC — Extended Binary Coded Decimal Interchange Code — exclusivo da IBM — grupo de 8 bits, permitindo a codificação de 256 símbolos diferentes.

ASCII — American Standard Code for Information Interchange — usado pelos demais fabricantes — grupo de 7 bits, sendo normalmente usado com mais 1 bit de paridade. Atualmente, com a necessidade de codificação de mais caracteres que os 128 possíveis com 7 bits (gráficos principalmente), há uma versão estendida do ASCII, com 8 bits, desenvolvida para aplicação com os microcomputadores de 16 bits (IBM-PC e compatíveis).

UNICODE — trata-se de um código de 16 bits por símbolo (portanto, pode representar um máximo de 65.536 símbolos diferentes) que pretende ser universal, ou seja, pode codificar em um único código símbolos de qualquer linguagem conhecida no mundo, solucionando vários problemas com os códigos atuais (na realidade há diversos conjuntos para o código ASCII devido ao fato de ele poder codificar apenas 256 símbolos diferentes), como a necessidade de várias versões para atender aos caracteres diferentes de várias linguagens (grego, hebraico, chinês e japonês, francês, inglês, português, espanhol etc.), além dos naturais conflitos decorrentes de essas versões produzirem, por exemplo, vários símbolos com um mesmo código. O Unicode está sendo desenvolvido por um consórcio constituído em 1991, compreendendo representantes das mais importantes empresas e órgãos governamentais.

Maiores informações sobre o Unicode podem ser encontradas em seu site na Web (www.unicode.org).

O Apêndice B apresenta as tabelas de símbolos codificados nos códigos ASCII e EBCDIC.

Na realidade, a forma de texto (conjunto de caracteres que possuem um código de representação válido para um certo sistema de computação) é o método primário de introdução de informações no computador (sejam essas instruções de um programa ou dados). As demais formas de representação de informação (tipos de dados) surgem no decorrer do processo de compilação ou interpretação do programa. Assim, ao introduzirmos, via teclado, o seguinte trecho de um programa em Basic:

```
DIM NOTA(12)

DATA 26.5, 45, 29

FOR I = 0 TO 12

READ NOTA(I)

NEXT I
```

o sistema converte cada caractere em um conjunto de pulsos elétricos, cada um com valores correspondentes ao bit 0 ou bit 1, cada conjunto indicando a representação de um dos caracteres digitados. Se, por exemplo, cada caractere for codificado com 8 bits (1 byte) e cada célula da MP armazenasse 1 byte, então cada caractere do programa citado seria armazenado em uma célula da memória do sistema usado, conforme mostrado na Fig. 7.2.

O valor decimal 26.5, por exemplo, é introduzido como caractere 2, caractere 6, caractere ponto (.) e caractere 5.

Quando o compilador ou interpretador Basic processar o programa, provavelmente converterá o valor 26.5 para representação denominada ponto flutuante (a notação científica mostrada anteriormente) (ver item 7.5.3).

E	C	E	C	E	C	E	C	E	C
70	01000100	7E	01000001	8C	00110010	9A	01001111	A8	00101000
71	01001001	7F	01010100	8D	00111001	9B	00100000	A9	01001001
72	01001101	80	01000001	8E	10001101	9C	00110001	AA	00101001
73	00100000	81	00100000	8F	01000110	9D	00110010	AB	10001101
74	01001110	82	00110010	90	01001111	9E	10001101	AC	01001110
75	01001111	83	00110110	91	01010010	9F	01010010	AD	01000101
76	01010100	84	00101110	92	00100000	A0	01000101	AE	01011000
77	01000001	85	00110101	93	01001001	A1	01000001	AF	01010100
78	00101000	86	00101100	94	00100000	A2	01000100	B0	00100000
79	00110001	87	00100000	95	00111101	A3	00100000	B1	01001001
7A	00110010	88	00110100	96	00100000	A4	01001110	B2	10001101
7B	00101001	89	00100101	97	00110000	A5	01001111		
7C	10001101	8A	00101100	98	00100000	A6	01010100		
7D	01000100	8B	00100000	99	01010100	A7	01000001		

E - Endereço da MP, em hexadecimal
C - Conteúdo da MP, célula por célula, em valores binários

Figura 7.2 Armazenamento em MP (binário) de um programa Basic.

Ou seja, primeiramente os caracteres são introduzidos um a um como um texto livre, sendo convertidos para o código de bits usado pela máquina; em seguida, ao serem traduzidos para código-objeto, os elementos do programa são passados para uma representação passível de ser interpretada e manipulada pelo hardware (pelo processador — UCP).

7.4 TIPO LÓGICO

Este tipo permite a utilização de variáveis que possuem apenas dois valores para representação, FALSO (usualmente representado pelo bit 0) e VERDADEIRO (representado pelo bit 1). Estas variáveis são utilizadas de diversas formas em um programa, inclusive podendo ser realizado um tipo específico de operação, empregando operadores lógicos (ver Cap. 4).

Em Pascal, é possível definir diversos tipos booleanos, distintos basicamente pelo tamanho da variável, como, por exemplo:

ByteBool — a variável irá ocupar 1 byte na memória; e

WordBool — a variável irá ocupar 1 palavra ou 2 bytes.

No Cap. 4 foram abordados, de forma detalhada, os aspectos essenciais relacionados às variáveis lógicas e aos diversos operadores lógicos encontrados em computadores. Foram ainda apresentados conceitos básicos de álgebra booleana, como também circuitos e elementos digitais. Porém, com o propósito de integrar o conceito de tipo de dado lógico ao ambiente deste capítulo, repetiremos aqui apenas alguns aspectos do assunto, aqueles referentes tão-somente aos operadores lógicos mais comuns e suas tabelas verdade. Para o entendimento de tipos de dados e aspectos básicos de operações lógicas basta conhecer as informações aqui contidas, deixando o Cap. 4 para aqueles que queiram se aprofundar mais no assunto.

7.4.1 Operador Lógico AND

O operador lógico AND é definido de modo que o resultado da operação com ele será VERDADE (representado no computador como o dígito binário 1) *se e somente se* todas as variáveis de entrada forem VERDADE (= 1). Isto é, se a variável A (1.° operando ou variável) **E** (AND) a variável B (2.° operando ou variável) forem VERDADEIRAS, então o resultado X também será VERDADEIRO; caso contrário (basta apenas uma das variáveis não ser verdade), o resultado será FALSO (= 0).

A tabela verdade da operação AND será:

Tabela Verdade da Operação Lógica AND		
A	B	X = A and B ou X = A · B
0	0	0
0	1	0
1	0	0
1	1	1

Exemplo 7.7

Se as variáveis lógicas A e B possuem os seguintes valores:

A = 0 e B = 1,

então o valor de X na expressão lógica X = A and B será X = 0. Isto porque pela tabela verdade da operação lógica AND teremos:

X = A and B (ou pode-se representar assim A · B ou AB, como na multiplicação algébrica) = 0 · 1 = 0.

Exemplo 7.8

Sejam A = 0 1 1 0 e B = 1 1 1 0. Obter o valor de X na expressão lógica: X = A · B.

Solução

A operação lógica é realizada na UAL bit a bit, como as operações aritméticas, usando-se a tabela verdade por bit. Assim, iniciando dos algarismos menos significativos (mais à direita) teremos:

0 · 0 = 0 1 · 1 = 1 1 · 1 = 1 0 · 1 = 0

A operação completa, mostrada de forma semelhante a uma operação aritmética, ficaria assim:

$$\begin{array}{r} 0110 \\ \text{and } 1110 \\ \hline 0110 \end{array}$$

Se usássemos a tabela verdade diretamente teríamos:

Tabela Verdade para a Operação X = A · B		
A	B	X = A · B
0	1	0
1	1	1
1	1	1
0	0	0

Resultado: X = 0 1 1 0

Exemplo 7.9

Sejam A = 1 1 0 0, B = 1 0 0 0 e C = 0 0 1 0. Obter o valor de X na expressão lógica: X = A · B · C

Solução

Calcula-se a expressão da seguinte forma: primeiro R = A · B e, em seguida, X = R · C, sempre bit a bit, como a seguir mostrado:

Para R = A · B, da direita para a esquerda:

0 · 0 = 0   0 · 0 = 0   1 · 0 = 0   1 · 1 = 1   e   R = 1 0 0 0

Para X = R · C, teremos:

0 · 0 = 0   0 · 1 = 0   0 · 0 = 0   1 · 0 = 0   e   X = 0 0 0 0

Se usássemos a tabela verdade diretamente teríamos que criar duas tabelas; uma para R = A · B e outra para X = R · C:

Tabela Verdade da Operação Lógica R = A · B		
A	B	R = A · B
1	1	1
1	0	0
0	0	0
0	0	0

Tabela Verdade da Operação Lógica X = R · C		
R	C	X = R · C
1	0	0
0	0	0
0	1	0
0	0	0

Resultado: X = 0 0 0 0

7.4.2 Operador Lógico OR

O operador lógico OR (OU) é definido de modo que o resultado da operação será VERDADE (= 1) se um operando (ou variável lógica) OU o outro for VERDADEIRO (basta que apenas um dos operandos seja VERDADEIRO); caso contrário, o resultado será FALSO (= 0).

A tabela verdade da operação OR será, então:

Tabela Verdade da Operação Lógica OR		
A	B	X = A or B ou X = A + B
0	0	0
0	1	1
1	0	1
1	1	1

O símbolo matemático para a operação lógica OR é o sinal de adição aritmética ("+"). Desse modo, representa-se a operação por uma das duas formas:

A or B   ou   A + B

Exemplo 7.10

Se as variáveis lógicas A e B possuem os seguintes valores:

A = 1 e B = 0,

então o valor de X na expressão lógica X = A or B será X = 1. Isto porque pela tabela verdade da operação lógica OR teremos:

X = A or B = A + B = 1 + 0 = 1.

Exemplo 7.11

Sejam A = 0 1 0 1 e B = 0 1 1 0. Obter o valor de X na expressão lógica: X = A + B.

Solução

A operação lógica é realizada na UAL bit a bit, como as operações aritméticas, usando-se a tabela verdade por bit. Assim, iniciando dos algarismos menos significativos (mais à direita) teremos:

1 + 0 = 1   0 + 1 = 1   1 + 1 = 1   0 + 0 = 0

A operação completa, mostrada de forma semelhante a uma operação aritmética, ficaria assim:

$$\begin{array}{r} 0101 \\ \underline{\text{or } 0110} \\ 0111 \end{array}$$

Se usássemos a tabela verdade diretamente teríamos:

Tabela Verdade para a Operação X = A + B		
A	B	X = A + B
0	0	0
1	1	1
0	1	1
1	0	1

Resultado: X = 0 1 1 1

Exemplo 7.12

Sejam A = 1 1 0 0, B = 1 0 0 0 e C = 0 0 1 0. Obter o valor de X na expressão lógica:

X = A + B + C

Solução

Calcula-se a expressão da seguinte forma: primeiro R = A + B e, em seguida, X = R + C, sempre bit a bit, como a seguir mostrado:

Para R = A + B, da direita para a esquerda:

0 + 0 = 0   0 + 0 = 0   1 + 0 = 1   1 + 1 = 1   e   R = 1 1 0 0

Para X = R + C, teremos:

0 + 0 = 0   0 + 1 = 1   1 + 0 = 1   1 + 0 = 1   e   X = 0 0 0 0

Se usássemos a tabela verdade diretamente teríamos que criar duas tabelas; uma para R = A + B e outra para X = R + C:

Tabela Verdade da Operação Lógica R = A + B		
A	B	R = A + B
1	1	1
1	0	1
0	0	0
0	0	0

Tabela Verdade da Operação Lógica X = R + C		
R	C	X = R + C
1	0	1
1	0	1
0	1	1
0	0	0

Resultado: X = 1 1 1 0

7.4.3 Operador Lógico NOT

O operador NOT, também conhecido como INVERSOR ou COMPLEMENTO (Inverter ou Complement), é definido de modo a produzir na saída um resultado de valor oposto (ou inverso) ao da variável de entrada. Naturalmente, em um sistema numeral onde somente há dois valores, 0 e 1, o inverso de um é o outro. Este operador é o único que pode ser utilizado sobre apenas uma única variável. Desse modo, se a variável tem o valor 0 (FALSO), o resultado da operação NOT sobre essa variável será 1 (VERDADE), e se a variável for igual a 1 (VERDADE), então o resultado do NOT será 0 (FALSO); sempre o valor oposto como resultado.

A tabela verdade da operação NOT será, então:

Tabela Verdade da Operação Lógica NOT	
A	X = NOT A ou X = $\overline{A}$
0	1
1	0

O símbolo matemático para a operação lógica OR é o traço superposto sobre a variável. Desse modo, representa-se a operação por uma das duas formas:

not A   ou $\overline{A}$

Exemplo 7.13

Se as variáveis lógicas A, B e C possuem os seguintes valores:

A = 1;   B = 101;   C = 1101

então o valor de X, Y e Z nas expressões lógicas X = not A, Y = not B e Z = not C será:

X = 0;   Y = 010   e   Z = 0010. Isto porque pela tabela verdade da operação lógica NOT o inverso de 0 é 1 e o inverso de 1 é 0. Neste exemplo foram invertidos os valores de cada bit das variáveis indicadas.

É também possível que, em vez de uma só variável, o resultado de uma expressão seja invertido, como também apenas parte da expressão.

Exemplo 7.14

Seja A = 1101. Calcule: X = NOT A ou X = $\overline{A}$.

Solução

Para calcular o valor de X, inverte-se cada bit de A e assim:

X = 0 0 1 0

Se usar diretamente a tabela verdade da operação NOT:

Tabela Verdade da Operação Lógica X = not A	
A	X = NOT A ou X = $\overline{A}$
1	0
1	0
0	1
1	0

Resultado: X = 0 0 1 0

É também possível que, em vez de uma só variável, o resultado de uma expressão seja invertido, como também pode-se inverter apenas parte da expressão.

Exemplo 7.15

Sejam as variáveis A = 011; B = 110; C = 001; D = 111. Calcular o valor de X na expressão:

$$X = \overline{D} \cdot (B + \overline{C \cdot A})$$

Solução

Na expressão indicada a execução das operações se realiza na seguinte ordem, de modo semelhante à da aritmética, ou seja: em primeiro lugar resolvem-se as expressões no interior dos parênteses, tendo prioridade a operação de AND (como a multiplicação sobre a soma) sobre a operação OR.

Uma diferença para a aritmética consiste no emprego, neste caso, da operação NOT (não existente na aritmética convencional), a qual é realizada em primeiro lugar, se a inversão for apenas de uma variável, ou após a realização da operação ou expressão se a inversão for de uma operação ou operações).

Dessa forma, a ordem de execução será: primeiro, a operação dentro dos parênteses, iniciando pelo AND de C e A, em seguida, invertendo o resultado e, depois, a operação deste resultado OR B. Finalmente, inverte-se o valor de D e executa-se a operação final AND.

1.ª oper.	2.ª oper.	3.ª oper.	4.ª oper.	5.ª oper.
001	not 001	110	not 111	000
and 011	110	or 110	000	and 110
001		110		000

7.4.4 Operador Lógico EXCLUSIVE-OR (OU EXCLUSIVO)

O operador lógico XOR (EXCLUSIVE-OR) ou OU EXCLUSIVO é definido de modo a prover um resultado VERDADEIRO se apenas uma das duas variáveis ou operadores for VERDADEIRA. Isto é: sendo X = A xor B, o resultado X será VERDADE se *exclusivamente* (daí o nome OU EXCLUSIVO) A OU B for VERDADE. Caso ambos sejam VERDADE ou ambos FALSOS, então o resultado será FALSO.

A diferença conceitual entre os operadores OR e XOR está no termo EXCLUSIVO; no caso do operador OR, as variáveis não necessitam ter valores *verdade* exclusivos, bastando apenas que ocorra um valor VERDADE (no entanto, os demais valores *também* podem ser VERDADE). No caso do operador XOR, isso não é possível: é preciso que uma variável ou outra seja VERDADE; não podem ambas ser VERDADE, pois deixa, nesse caso, de haver exclusividade.

A tabela verdade da operação XOR será então:

Tabela Verdade da Operação Lógica XOR		
A	B	X = A xor B ou X = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

O símbolo matemático para a operação lógica XOR é o sinal de adição aritmética ("⊕") envolvido por um círculo. Desse modo, representa-se a operação por uma das duas formas:

A or B ou A ⊕ B

Exemplo 7.16

Se as variáveis lógicas A e B possuem os seguintes valores:

$A = 1$ e $B = 0$,

então o valor de X na expressão lógica $X = A \oplus B$ será $X = 1$. Isto porque pela tabela verdade da operação lógica XOR teremos:

$X = A \text{ xor } B = A \oplus B = 1 \oplus 0 = 1$.

Exemplo 7.17

Sejam $A = 1\ 1\ 0\ 1$ e $B = 1\ 0\ 0\ 0$. Obter o valor de X na expressão lógica: $X = A \oplus B$.

Solução

A operação lógica é realizada na UAL bit a bit, como as operações aritméticas, usando-se a tabela verdade por bit. Assim, iniciando dos algarismos menos significativos (mais à direita) teremos:

$1 \oplus 0 = 1 \qquad 0 \oplus 0 = 0 \qquad 1 \oplus 0 = 1 \qquad 1 \oplus 1 = 0$

A operação completa, mostrada de forma semelhante a uma operação aritmética, ficaria assim:

$$\begin{array}{r} 1101 \\ \text{or}\ \ 1000 \\ \hline 0101 \end{array}$$

Se usássemos a tabela verdade diretamente teríamos:

Tabela Verdade para a Operação $X = A \oplus B$		
A	B	$X = A \oplus B$
1	1	0
1	0	1
0	0	0
1	0	1

Resultado: $X = 0\ 1\ 0\ 1$

Exemplo 7.18

Sejam $A = 1\ 1\ 0\ 0$, $B = 1\ 0\ 0\ 0$ e $C = 0\ 0\ 1\ 0$. Obter o valor de X na expressão lógica: $X = A \text{ xor } B \text{ xor } C$ ou $X = A \oplus B \oplus C$.

Solução

Calcula-se a expressão da seguinte forma: primeiro $R = A \oplus B$ e, em seguida, $X = R \oplus C$, sempre bit a bit, como a seguir mostrado:

Para $R = A \oplus B$, da direita para a esquerda:

$0 \oplus 0 = 0 \qquad 0 \oplus 0 = 0 \qquad 1 \oplus 0 = 1 \qquad 1 \oplus 1 = 0$ e $R = 0\ 1\ 0\ 0$

Para $X = R \oplus C$, teremos:

$0 \oplus 0 = 0 \qquad 0 \oplus 1 = 1 \qquad 1 \oplus 0 = 1 \qquad 0 \oplus 0 = 0$ e $X = 0\ 1\ 1\ 0$

Se usássemos a tabela verdade diretamente teríamos que criar duas tabelas; uma para $R = A \oplus B$ e outra para $X = R \oplus C$:

Tabela Verdade da Operação Lógica $R = A \oplus B$		
A	B	$R = A \oplus B$
1	1	0
1	0	1
0	0	0
0	0	0

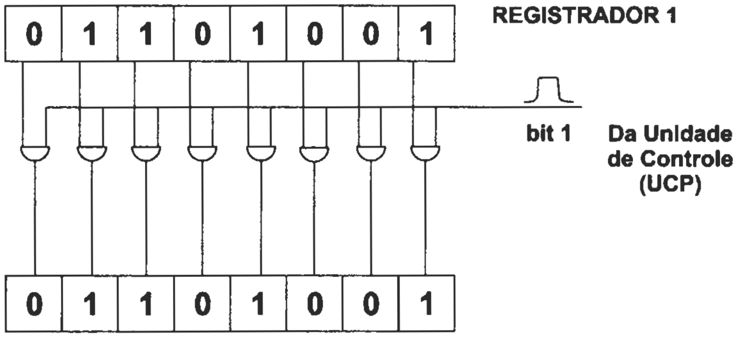
Tabela Verdade da Operação Lógica $X = R \oplus C$		
R	C	$X = R \oplus C$
0	0	0
1	0	1
0	1	1
0	0	0

Resultado: X = 0 1 1 0

As operações lógicas mostradas são parte do conjunto de operações existentes na álgebra lógica, conforme já descrito com mais detalhe no Cap. 4. Essas operações possuem características marçantes nos sistemas de computação. Assim, o operador lógico AND serve como "porta" ou "gate" de passagem de bits; ou seja, um determinado conjunto de bits de um registrador ou célula de memória pode ser transferido para outro registrador ou célula (como acontece freqüentemente durante a realização dos ciclos de memória ou de execução de instruções, durante o instante de tempo (nanossegundos) em que um pulso de controle VERDADE se combina com cada sinal elétrico armazenado do elemento de origem através de um circuito lógico AND). O resultado de saída desta combinação será igual ao valor de origem, isto é, transferindo o valor de origem para a saída. Em resumo:

A AND 1 = A (ver Fig. 7.3).

O registrador 1 armazena o valor binário: 01101001. Deseja-se transferir uma cópia deste valor para o registrador 2, ou até mesmo para a célula da MP. Para uso, um sinal de controle (pulso = bit 1) é inserido na porta juntamente com o valor do registrador 1. A saída de cada porta AND irá para o registrador 2.



O bit 1 (pulso controle), combinado em cada porta AND com o valor do bit correspondente no registrador 1, acarreta uma saída para o registrador 2 que é de valor binário igual ao do registrador 1, pois A AND 1 = A.

Figura 7.3 Exemplo da utilização da porta lógica AND.

O operador lógico XOR também tem um papel importante nos sistemas de computação em virtude de sua característica de produzir um resultado VERDADE, se os valores de entrada forem diferentes, e FALSO, se os valores de entrada forem iguais, ou seja:

0 xor 1 e 1 xor 0 produzem sempre resultado VERDADE
0 xor 0 e 1 xor 1 produzem sempre resultado FALSO.

Esta propriedade permite que se possa verificar a igualdade ou não entre dois valores (muito necessário em operações aritméticas para verificação da igualdade ou não do sinal entre dois números).

7.5 TIPO NUMÉRICO

Como os computadores são elementos binários, a forma mais eficiente de representar números deve ser a binária, isto é, converter o número diretamente de decimal para seu correspondente valor binário. A unidade aritmética e lógica (UAL) dos computadores executa operações mais rapidamente se os valores estiverem representados desse modo.

Para se trabalhar em computação com valores numéricos deve-se levar em consideração três fatos que podem acarretar inconvenientes no projeto e na utilização da máquina e que, na nossa vida cotidiana (aritmética com papel e lápis), não causam nenhum problema:

- a representação do sinal de um número;
- a representação da vírgula (ou ponto) que separa a parte inteira da parte fracionária de um número não-inteiro; e
- a quantidade limite de algarismos possível de ser processada pela UAL de um processador.

O primeiro dos problemas, que consiste na indicação do sinal do número, é resolvido com o acréscimo de mais um bit na representação do número. Esse bit adicional indica o sinal do número. A convenção adotada de forma universal é:

- valor positivo: bit de sinal igual a zero;
- valor negativo: bit de sinal igual a um.

A Fig. 7.4 apresenta alguns exemplos de números binários representados com sinal.

Valor decimal	Valor binário Bit de sinal ↓	
+12	01100	(com 5 bits)
−12	11100	(com 5 bits)
−47	100101111	(com 9 bits)
+47	000101111	(com 9 bits)

Figura 7.4 Exemplo de números com sinal.

Como podemos observar, no computador toda e qualquer informação somente é representada sob a forma de 0s ou 1s, diferentemente de nossa linguagem (dos humanos), na qual utilizamos vários símbolos diferentes, para diferentes itens, como os símbolos visualmente diferentes de um valor negativo ("−"), bit 1 em computação ou positivo ("+"), bit 0. Em geral, o bit de sinal é inserido à esquerda do número, como bit mais significativo.

Algumas das linguagens de programação permitem o emprego de valores com sinal e sem sinal. Neste último caso, as operações aritméticas são mais simples, pois não precisam considerar o valor do sinal do número.

O segundo problema reside na forma de representação de números fracionários. Isso ocorre devido à dificuldade de representar-se a vírgula (ou ponto de separação entre a parte inteira e fracionária do número) internamente, entre a posição de dois bits, mais uma vez, devido ao fato de que não existe símbolo[2] para a vírgula (",") em computação. Na realidade, a vírgula não é efetivamente representada, mas sim assumida sua posição no número e este sendo representado apenas pelos seus algarismos significativos como se fosse inteiro. Ou seja, o número 110111,110 seria representado internamente como 110111110 e o sistema "saberia" que os três últimos algarismos à direita seriam fracionários.

O "saber" que quantidade de algarismos é inteira e que quantidade é fracionária é um problema cuja solução pode ser encontrada pela escolha entre dois modos de representação e de realização de operações aritméticas:

- representação em ponto fixo (ou vírgula fixa), usualmente indicada e usada para valores inteiros;
- representação em ponto flutuante (vírgula flutuante), usada para valores fracionários — números reais.

De modo geral, os sistemas de computação somente representam a vírgula de separação entre as partes inteira e fracionária dos números (ou ponto fracionário, como adotado nos EUA) quando o número é introduzido no sistema como um texto livre. A Fig. 7.5 mostra um exemplo de representação de um número pela codificação de cada um de seus algarismos no código ASCII.

$+37{,}5_{10}$	00101011	00110011	00110111	00101100	00110101
	+	3	7		5

Figura 7.5 Representação de um número em código ASCII.

Quando o número é convertido para uma forma binária pura, onde sua magnitude é um valor binário correspondente ao decimal de entrada, então a vírgula fracionária (ou ponto) é assumida em uma determinada posição (definida na declaração do tipo da variável efetuada no corpo do programa-fonte); não há espaço nas células de memória para armazenar um bit de vírgula.

Finalmente, o terceiro fato — quantidade de algarismos disponível no sistema de computação para representar números — já mencionado anteriormente, no item 7.1, é realmente um problema, visto que, na nossa matemática comum, este fato não existe. Em outras palavras, na matemática a quantidade de números reais existente é infinita; podemos ter infinitos valores entre qualquer faixa de números (dependendo, é claro, do tamanho do papel que temos disponível). Por exemplo, há infinitos valores entre 7,01 e 7,02, tais como 7,011, 7,012, 7,0123, 7,012245, e assim por diante.

No entanto, computadores são máquinas de tamanho finito, elementos finitos, células e registradores de tamanho finito e, por causa disso, somente têm capacidade de representar uma quantidade finita de números. Isso acarreta um problema de precisão e erros tanto para representar números quanto na ocasião de obter-se o resultado de operações aritméticas. Desse modo surge o aspecto denominado *overflow* ou estouro da capacidade de representar números que a natureza finita da máquina produz. Além do *overflow*, que é o excesso do limite superior de representação, há também *underflow*, que é o excesso para menos (o resultado é um valor menor que o menor valor representável com uma determinada quantidade de algarismos, disponível em uma dada máquina).

A questão do erro em operações aritméticas devido ao excesso da quantidade de algarismos pode ocorrer de muitas e perigosas maneiras, como se pode ver pelo exemplo a seguir.

Exemplo 7.19

Considere-se os seguintes números decimais com 2 algarismos: A = 45; B = 63 e C = −53.

Primeiramente, vamos considerar a seguinte operação: A + (B + C).

---

[2] Naturalmente os códigos de representação de caracteres possuem um código para a vírgula, como também para o ponto, dois pontos etc. Não estamos aqui nos referindo a este fato, verdadeiro, mas sim ao fato de que em um número binário não há um bit para indicar a posição da vírgula.

A operação seria: 45 + (63 − 53) = 45 + 10 = 55.

No entanto, se resolvermos realizar esta outra operação com os mesmos dados, o resultado não poderá ser obtido de forma correta, senão vejamos:

(A + B) + C = (45 + 63) − 53 = 108 − 53 = 55

Considerando-se apenas as leis da matemática e o fato de essas operações estarem sendo realizadas no papel, onde não há excesso de limite da quantidade permitida de algarismos de cada número, ambos os resultados serão iguais a 55, sem qualquer problema. No entanto, se considerarmos que as operações estão sendo realizadas por um computador, onde cada número é representado apenas por 2 algarismos, então a segunda operação estará incorreta, pois ocorreu *overflow* no resultado intermediário de A + B = 108, visto que 108 é um número de 3 algarismos. Trata-se, portanto, de um problema que pode ocorrer com certa freqüência nos sistemas computacionais, como veremos mais adiante.

No item 7.5.2 trataremos este assunto com um pouco mais de detalhes.

7.5.1 Representação em Ponto Fixo

Esse método consiste na determinação de uma posição fixa para a vírgula (ou ponto). Todos os valores representados em ponto fixo para uma determinada operação possuem a mesma quantidade de algarismos inteiros, bem como a mesma quantidade de algarismos fracionários. Por exemplo:

1101,101 1110,001 0011,110

As posições mais adotadas para a vírgula são:

- na extremidade esquerda do número — nesse caso, o número é totalmente fracionário;
- na extremidade direita do número — nesse caso, o número é inteiro.

Em qualquer desses casos, no entanto, a vírgula fracionária não estará fisicamente representada na memória; sua posição é determinada na definição da variável, realizada pelo programador (ou pelo compilador), e o sistema memoriza essa posição, mas não a representa fisicamente.

Na quase totalidade das linguagens de programação e nos sistemas de computação (e os compiladores da maior parte das linguagens de programação) emprega-se a representação de números em ponto fixo para indicar apenas valores inteiros (a vírgula fracionária é assumida na posição mais à direita do número); números fracionários são, nesses casos, representados apenas em ponto flutuante (ver item 7.5.3).

A Fig. 7.6 mostra alguns exemplos de declarações de dados, onde se observa a palavra-chave da linguagem (indica o tipo de dados para o compilador) e a correspondente representação utilizada pelo compilador para o armazenamento e as operações aritméticas a serem eventualmente realizadas com o referido dado.

Linguagem	Tipos de dados	Representação interna
Basic	INTEGER	Ponto fixo (inteiro c/ 16 bits)
	SINGLE-PRECISION	Ponto flutuante (real c/ 32 bits)
	DOUBLE-PRECISION	Ponto flutuante (real c/ 64 bits)
Fortran	INTEGER	Ponto fixo (inteiro)
	REAL	Ponto flutuante (real)
Cobol	COMP	Ponto fixo (inteiro)
	COMP 1	Ponto flutuante (real)
	COMP 2	Ponto flutante (real)
	COMP 3	Decimal compactado (decimal)
Pascal	INTEGER	Ponto fixo (inteiro)
	REAL	Ponto flutuante (real)

Figura 7.6 Exemplos de tipos de dados em algumas linguagens.

Na representação de números em ponto fixo, os valores positivos são representados pelo bit zero de sinal. Normalmente esse bit é posicionado como o algarismo mais significativo do número, o mais à esquerda, e pelos n-1 restantes bits, que correspondem ao valor absoluto do número (magnitude).

Quanto aos números negativos, o sinal é representado pelo bit um e a magnitude pode ser representada por um dos três seguintes modos:

- sinal e magnitude;
- complemento a 1;
- complemento a 2.

Conforme verificaremos a seguir, as operações com números negativos (soma com uma das parcelas negativa ou subtração de números) são difíceis e demoradas, quando realizadas no modo sinal e magnitude. Isso porque é necessário efetuar várias comparações e decisões, em vista da manipulação dos sinais das parcelas para determinação do sinal do resultado.

Também mostraremos que as mesmas operações se tornam mais simples e rápidas se realizadas através da aritmética de complemento. Essa é a razão básica do seu emprego em computadores digitais (ver item 7.5.1.2).

Em cada um dos três métodos não só mostraremos como os números são representados, mas também descreveremos um algoritmo básico utilizado para realizar operações de soma e subtração em cada caso.

7.5.1.1 Sinal e Magnitude

A representação de números com *n* algarismos binários (*n* bits) em sinal e magnitude é obtida atribuindo-se 1 bit (em geral, na posição mais à esquerda do número) para indicar o valor do sinal, e os n-1 bits restantes para indicarem a magnitude (a grandeza) do número, como mostrado na Fig. 7.7 (o método de posicionamento dos bits é próprio da representação em ponto fixo e, portanto, será também igual em complemento a 1 e complemento a 2).

Nesse tipo de representação, o valor dos bits usados para representar a magnitude (seu valor absoluto) do número é o mesmo, seja o número positivo ou negativo; o que varia é apenas o valor do bit de sinal, exatamente como acontece quando usamos os símbolos de nossa linguagem, com a diferença de que usamos símbolos gráficos diferentes dos bits do computador (ver Fig. 7.8).

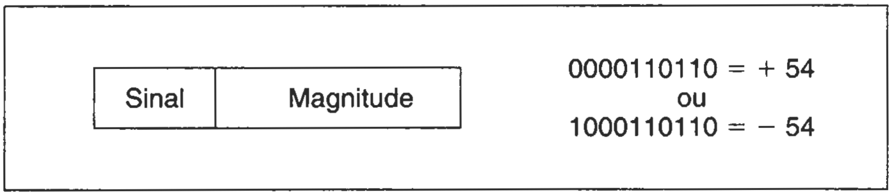


Figura 7.7 Exemplo de uma representação de dado em sinal e magnitude.

Para representar internamente cada número, a máquina converte o valor absoluto (a magnitude) do número (que deverá estar representado em base decimal) para seu valor correspondente em base 2 e acrescenta um bit à esquerda (que passa a ser o algarismo mais significativo), cujo valor será igual a 0 se o número for positivo (+) ou igual a 1 se o número for negativo (−).

Limites de Representação

Se os registradores que irão armazenar os valores possuem capacidade para receber *n* algarismos, então a faixa-limite de números inteiros, que pode ser armazenada nos referidos registradores, é obtida da expressão (7.1):

$$\boxed{-(2^{n-1} - 1) \quad \text{a} \quad +(2^{n-1} - 1)} \qquad (7.1)$$

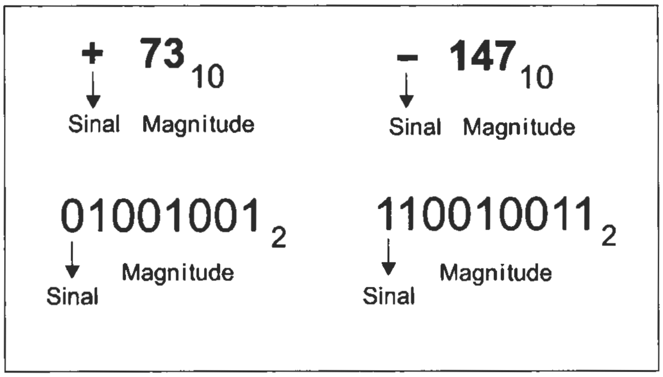


Figura 7.8 Representação de valores em serial e magnitude.

Se *n* é a quantidade-limite de algarismos, então a magnitude é calculada a partir de n-1 algarismos, visto que 1 algarismo é reservado para indicar o sinal do número.

Do valor obtido (2^{n-1}) subtrai-se 1 (para os valores negativos e positivos) porque o primeiro valor a ser representado é o valor zero (0).

Ainda sobre o modo de representação sinal e magnitude (ver Tabela 7.2), a Fig. 7.9 apresenta alguns exemplos da representação binária em sinal e magnitude.

A referida representação possui algumas características que, comparadas com as demais representações em ponto fixo (complemento a 1 e complemento a 2), tornam-na menos vantajosa que as outras, razão por que não é utilizada atualmente nos processadores. São elas:

a) Possui duas representações para o zero (matematicamente incorreto), o que é uma desvantagem em relação ao modo complemento a 2. Essas representações são o valor zero precedido do bit 0 (positivo) e o valor zero precedido do bit 1 (negativo).

   $+0_{10} = 0000_2$ e $-0_{10} = 1000_2$

b) A representação de números é simétrica entre positivos e negativos, limitada à quantidade permitida de bits dos registradores internos. Ou seja, se temos um registrador com capacidade para armazenar 6 bits, podem ser introduzidos nesse registrador até 64 combinações de valores binários, pois $2^6 = 64$.

Essas representações seriam desde 000000 até 111111, sendo divididas em duas partes de 32 representações cada uma. A primeira parte consistiria em números iniciados por 0, cuja faixa seria 000000 a 011111 (valores positivos, de decimal 0 até decimal 31) e a outra faixa, de 32 valores negativos, de 100000 a 111111 (valores negativos, de decimal 0 até decimal 31).

c) Conforme se observa no exemplo da Fig. 7.9, se os números são fracionários, a faixa de representação é mais reduzida, porque o valor n-1 refere-se tão-somente à quantidade de algarismos da parte inteira.

Aritmética em Sinal e Magnitude

Em sistemas de computação, o sinal é um símbolo de forma idêntica à dos algarismos representativos do seu valor (é um bit igual aos demais), diferentemente da linguagem humana, onde o sinal (= ou −) é um símbolo de forma diversa daquela utilizada para os algarismos que representam o valor do número (0, 1, 2, 5 ...). Além disso, na forma dos humanos, ele é colocado separado da magnitude (+31), não fazendo parte dos cálculos em si, mas tão-somente servindo para definir o sinal do resultado e o tipo da operação a ser realizada efetivamente, enquanto, nos computadores, o bit de sinal está incorporado aos bits da magnitude, formando um número só.

a) Para um registrador de 6 bits, n = 6, e os limites de representação serão:

de $-(2^{6-1}-1)$ a $+(2^{6-1}-1)$ ou
$-(2^5-1)$ a $+(2^5-1) = -31$ a $+31$

b) Para um registrador de 16 bits, n = 16, e teremos:

de $-(2^{16-1}-1)$ a $+(2^{16-1}-1) = -32.767$ a $+32.767$

c) Para um registrador de 16 bits, porém representando um número fracionário tendo, por exemplo, 10 bits para a parte inteira, 5 bits para a parte fracionária e 1 bit para o sinal.

Nesse caso, n = 10 (somente a parte inteira, é claro) e os limites serão:

de $-(2^{10-1}-1)$ a $(2^{10-1}-1) = -(2^9-1)$ a $+(2^9-1) = -511$ a $+511$

Figura 7.9 Exemplos de limites de representação.

Aparentemente, parece que a maneira mais simples de efetuar uma soma seria considerar o número como um inteiro sem sinal; caso contrário, o resultado será incorreto:

$$\begin{array}{rl} & 0100\ +4 \\ + & 1010\ -2 \\ \hline & 1110\ -6 \end{array} \qquad \begin{array}{rl} & 0110\ +2 \\ + & 1010\ -2 \\ \hline & 1100\ -4 \end{array}$$

incorreto incorreto

Esses exemplos mostram claramente que o bit de sinal não pode, pelo menos nessa forma de representação (sinal e magnitude), ser considerado na operação; ele será necessário apenas para identificar o tipo da operação e o sinal do resultado, conforme já mencionado anteriormente.

Por isso, as operações aritméticas em sinal e magnitude são efetuadas de modo idêntico ao que fazemos com lápis e papel, isto é, informalmente, executamos um algoritmo para determinar o sinal do resultado e, depois, efetuamos a operação propriamente dita, utilizando apenas as magnitudes.

Assim, se o leitor em seu dia-a-dia se defrontar com a necessidade de efetuar a operação

13 − 17

prontamente (acredito mentalmente) obterá como resultado o valor −4.

Na realidade, mentalmente o leitor executou o seguinte algoritmo:

1) Comparou os sinais dos números, + para 13 e − para 17, verificando que eram diferentes.
2) Deduziu que, sendo os sinais diferentes, a operação a ser realizada era de subtração.
3) Sendo operação de subtração, determinou o maior dos dois números (17) de modo que a operação será subtrair o menor do maior dos números (17 − 13) e que o resultado da operação terá o sinal do maior dos números ("−").
4) Desse modo, obteve −4 = −(17 − 13).

Para formalizar, são descritos em seguida os passos de um possível algoritmo para operações aritméticas de soma e subtração, utilizando-se números representados em sinal e magnitude:

Soma

1) Verificam-se os sinais dos números e efetua-se uma comparação entre eles.
2) Se ambos os números têm o mesmo sinal, somam-se as magnitudes; o sinal do resultado é o mesmo das parcelas.
3) Se os números têm sinais diferentes:
   a) identifica-se a maior das magnitudes e registra-se o seu sinal;
   b) subtrai-se a magnitude menor da maior (apenas as magnitudes);
   c) sinal do resultado é igual ao sinal da maior magnitude.

Exemplo 7.20

Utilizando-se o algoritmo de soma em sinal e magnitude, efetuar a soma dos números (+13) e (+12). Considerar a palavra de dados (tamanho dos registradores e da UAL) com 6 bits.

Solução

Comparando-se os sinais verifica-se que ambos são iguais (bit 0 em ambos). Portanto, somam-se as magnitudes e coloca-se como sinal do resultado o mesmo sinal dos números, isto é: bit 0, conforme item 2 do algoritmo.

+13	001101
+12	001100
+25	011001

Exemplo 7.21

Efetuar a soma de (−17) e (−9). Considerar a palavra de dados (tamanho dos registradores e da UAL) com 6 bits.

Solução

Trata-se de situação semelhante à do exemplo anterior, apenas com a diferença do valor do sinal.

Comparando-se os sinais verifica-se que ambos são iguais (bit 1 em ambos). Portanto, somam-se as magnitudes e coloca-se como sinal do resultado o mesmo sinal dos números, isto é: bit 1, conforme item 2 do algoritmo.

−17	110001
− 9	101001
−26	111010

Exemplo 7.22

Efetuar a soma de (+18) e (−11). Considerar a palavra de dados (tamanho dos registradores e da UAL) com 6 bits.

Solução

Comparando-se os sinais verifica-se que são diferentes e, portanto, recai-se no item 3 do algoritmo. A magnitude de maior valor é 18 (item 3a do algoritmo) e seu sinal é positivo (bit 0); subtrai-se 18 − 11 = 7 (item 3b); sinal do resultado será positivo (bit 0), porque é o sinal da maior magnitude (item 3c).

+18	010010
−11	101011
+ 7	000111

Exemplo 7.23

Somar (−21) e (+10). Considerar a palavra de dados (tamanho dos registradores e da UAL) com 6 bits.

Solução

Trata-se de caso semelhante ao do exercício anterior, apenas trocando-se o sinal que será atribuído ao resultado.

Comparando-se os sinais verifica-se que são diferentes e, portanto, recai-se no item 3 do algoritmo. A magnitude de maior valor é 21 (item 3a do algoritmo) e seu sinal é negativo (bit 1); subtrai-se 21 − 10 = 7 (item 3b); sinal do resultado será negativo (bit 1), porque é o sinal da maior magnitude (item 3c).

$$\begin{array}{rl} -21 & 110101 \\ +10 & 001010 \\ \hline -11 & 101011 \end{array}$$

Exemplo 7.24

Somar (−21) e (+10). Considerar a palavra de dados (tamanho dos registradores e da UAL) com 16 bits.

Solução

Trata-se de operação exatamente igual à do exercício anterior. A única diferença, para o leitor compreender melhor a questão da quantidade de bits de cada valor, reside no fato de que, neste exemplo, a palavra é de 16 bits e não mais de 6 bits. O leitor deve observar os valores dos algarismos.

$$\begin{array}{rl} -21 & 1\ 000000000010101 \\ +10 & 0\ 000000000001010 \\ \hline -11 & 1\ 000000000001011 \end{array}$$

Exemplo 7.25

Somar (−17) e (−19). Considerar a palavra de dados (tamanho dos registradores e da UAL) com 6 bits.

Neste exemplo iremos observar a ocorrência de um *overflow* ou estouro da capacidade dos registradores, visto que, tendo a palavra 6 bits, e considerando-se a expressão 7.1, da faixa de representação de valores em sinal e magnitude, identificamos como limites os valores −31 a +31. Como o resultado desta operação será igual a −36, este valor não poderá ser representado com 6 bits. Seria necessário que o sistema de computação possuísse palavra de, pelo menos, 7 bits, pois, neste caso, os limites passariam a ser de −63 a +63.

O estouro (*overflow*) é determinado quando ocorre a existência de um "vai 1" para o bit de sinal.

$$\begin{array}{rll} & 1\ 011 & \leftarrow \text{“Vai um” para o bit de sinal indica que houve estouro (\textit{overflow})} \\ -17 & 110001 & \\ -19 & 110011 & \\ \hline & 100100 & \end{array}$$

Subtração (Minuendo − Subtraendo = Resultado)

O algoritmo para o caso de operação de subtração é praticamente igual ao caso da soma, exceto que se realiza um passo anterior: troca-se o sinal do subtraendo. Daí em diante segue-se exatamente o algoritmo indicado para a operação de soma.

1) Troca-se o sinal do subtraendo.
2) Procede-se como no algoritmo da soma.

Exemplo 7.26

Efetuar a subtração: (−18) − (+12). Considerar palavra de dados com 6 bits.

Solução

Seguindo o algoritmo mostrado, troca-se o sinal do subtraendo (o valor atual é +12 e passa a ser −12). Em binário teríamos o seguinte, com palavra de 6 bits:

+12 = 0 01100 e −12 = 1 01100

Em seguida, executa-se o algoritmo definido para a operação de soma, ou seja: comparam-se os sinais dos números, verificando-se que são iguais (ambos são negativos — bit 1). Neste caso, somam-se os números e mantém-se o mesmo sinal (bit 1) para o resultado encontrado.

$$\begin{array}{rr} -18 & 1\ 10010 \\ -12 & 1\ 01100 \\ \hline -30 & 1\ 11110 \end{array}$$

Exemplo 7.27

Efetuar a subtração: (−27) − (−14). Considerar palavra de dados com 6 bits.

Solução

Seguindo o algoritmo, troca-se o sinal do subtraendo (passa de −14 para +14) e executa-se o algoritmo definido para somas.

Com palavra de 6 bits teremos para o subtraendo:

−14 = 1 01110 e +14 = 0 01110

Executando-se o algoritmo definido para a operação de soma, teremos: comparam-se os sinais dos números, verificando-se que são diferentes (27 é negativo — bit 1 e 14 é positivo — bit 0). Neste caso, a operação a ser realizada é de subtração (item 3); subtrai-se 14 de 27 e o sinal do resultado será negativo — bit 1, que é o sinal do maior dos números.

$$\begin{array}{rr} -27 & 1\ 11011 \\ +14 & 0\ 01110 \\ \hline -13 & 1\ 01101 \end{array}$$

Exemplo 7.28

Efetuar a subtração: (+27) − (+31). Considerar palavra de dados com 6 bits.

Solução

Seguindo o algoritmo, troca-se o sinal do subtraendo (passa de +31 para −31) e executa-se o algoritmo definido para somas.

Com palavra de 6 bits teremos para o subtraendo:

+31 = 0 11111 e −31 = 1 11111

Executando-se o algoritmo definido para a operação de soma, teremos: comparam-se os sinais dos números, verificando-se que são diferentes (27 é positivo — bit 0 e 31 é negativo — bit 1). Neste caso, a operação a ser realizada é de subtração (item 3); subtrai-se 27 de 31 e o sinal do resultado será negativo — bit 1, que é o sinal do maior dos números.

$$\begin{array}{rr} +27 & 0\ 11011 \\ -31 & 1\ 11111 \\ \hline -4 & 1\ 00100 \end{array}$$

Observação: na realidade a colocação das parcelas em uma operação de subtração usando-se papel seria iniciada, em cima, com o valor maior (no caso é 31), subtraindo-se do menor (27 neste exemplo), e não como está posicionado na figura, com o 27 na parte superior. Mantivemos apenas a ordem do enunciado do exemplo.

Exemplo 7.29

Efetuar a subtração: (+19) − (−25). Considerar palavra de dados com 6 bits.

Solução

Neste exemplo os valores escolhidos nos permitirão observar a ocorrência de *overflow* (estouro da capacidade dos registradores). Isto porque, com 6 bits, a faixa de representação de números é de −31 a +31 e o resultado da operação será −44, o que somente poderá ser representado com sinal se a palavra fosse, pelo menos, igual a 7 bits.

Seguindo o algoritmo, troca-se o sinal do subtraendo (passa de −25 para +25) e executa-se o algoritmo definido para somas.

Com palavra de 6 bits teremos para o subtraendo:

−25 = 1 11001 e +25 = 0 11001

Executando-se o algoritmo definido para a operação de soma, teremos: comparam-se os sinais dos números, verificando-se que são diferentes (27 é positivo – bit 0 e 31 é negativo – bit 1). Neste caso, a operação a ser realizada é de subtração (item 3); subtrai-se 27 de 31 e o sinal do resultado será negativo – bit 1, que é o sinal do maior dos números.

$$\begin{array}{rr} & 1\ \ 11 \\ +19 & 0\ 10011 \\ +25 & 0\ 11001 \\ \hline & 0\ 00100 \end{array}$$

Ocorreu "vai um" para o bit de sinal e, por isso, o resultado está incorreto.

O problema encontrado pelos fabricantes de computadores na implementação da UAL que efetuasse operações aritméticas com valores representados em sinal e magnitude residiu, principalmente, em dois fatores: *custo* e *velocidade*.

Custo, devido à necessidade de construção de dois elementos diferentes, um para efetuar somas e outro para efetuar subtrações (dois componentes eletrônicos custam mais caro do que um), e *velocidade*, ocasionada pela perda de tempo gasto na manipulação dos sinais, de modo a determinar o tipo da operação e o sinal do resultado.

Além disso, há também a inconveniência da dupla representação para o zero, o que requer um circuito lógico específico para evitar erros de má interpretação. O sistema pode ser programado, por exemplo, para executar uma determinada ação se e somente se o resultado de uma operação aritmética for igual a zero. Para realizar essa verificação é feito um teste do sinal do resultado que, sendo −0, bit de sinal igual a 1, redundaria, erroneamente, na não execução da ação desejada.

Embora atualmente a construção de circuitos lógicos complexos seja bastante barata e, considerando também que é desprezível o custo de fabricação de um circuito lógico para efetuar uma operação de subtração em uma UAL, nenhum sistema moderno emprega aritmética em sinal e magnitude, a qual foi definitivamente substituída pela aritmética em complemento a 2 (no caso, é claro, de representação em ponto fixo), cuja descrição será efetuada a seguir.

7.5.1.2 Representação de Números Negativos em Complemento

Em face dos inconvenientes apresentados pela representação e aritmética em sinal e magnitude e das vantagens que, em contrapartida, a representação em complemento a 2 (C2) possui, os sistemas de computação empregam, de modo generalizado, aritmética de complemento a 2 em vez de sinal e magnitude.

Neste item, serão apresentados os conceitos e detalhes da aritmética tanto em complemento a 2 (C2) quanto em complemento a 1 (C1), embora este último não seja empregado nos sistemas atuais.

A comparação entre os três modos de representação e aritmética para números em ponto fixo indica vantagens de C2 e C1 sobre sinal e magnitude e mais vantagens ainda na aritmética e representação de C2 sobre C1. A Tabela 7.1 mostra um resumo comparativo entre as três modalidades de representação para ponto fixo.

Tabela 7.1 Quadro Comparativo entre as Modalidades de Representação em Ponto Fixo

Tipo de representação	Dupla representação para o zero	Custo	Velocidade
Sinal e magnitude	SIM (desvantagem)	Alto (componentes separados para soma e subtração)	Baixa (algoritmo de verif. sinais, soma e subtração)
Complemento a 1	SIM (desvantagem)	Baixo (um componente único para soma e subtração)	Média (operação mais demorada que em C2)
Complemento a 2	NÃO (vantagem)	Baixo (um componente único para soma e subtração)	Alta (algoritmo simples e igual para soma e subtração)

O conceito matemático de complemento é válido para qualquer base de numeração (B). Há dois tipos de complemento: *complemento à base (C a B)* e *complemento à base menos um (C a B − 1).*

Substituindo B pelo valor de uma determinada base, podemos ter, por exemplo:

- complemento a 10 e complemento a 9 (se a base B = 10);
- complemento a 2 e complemento a 1 (se a base B = 2);
- complemento a 16 e complemento a 15 (se a base B = 16 ou hexadecimal), e assim por diante.

Complemento à Base

Em matemática, o termo *complemento* significa a quantidade que falta para "completar" um valor, torná-lo completo.

Por exemplo, o complemento de um ângulo agudo é o valor de graus que precisa ser adicionado ao ângulo para se obter 90°, considerando-se, nesse caso, que um ângulo de 90° é completo.

Em operações aritméticas, o complemento à base de um número N é o valor necessário para se obter B^n, ou seja:

complemento à base de $N = B^n - N$, onde
n = quantidade de algarismos utilizados na operação; e
N = valor do número.

Tabela 7.2 Correspondência entre Valores Decimais e nas Representações em Ponto Fixo

Decimal	Sinal e magnitude	Complemento a 1	Complemento a 2
−16	—	—	10000
−15	11111	10000	10001
−14	11110	10001	10010
−13	11101	10010	10011
−12	11100	10011	10100
−11	11011	10100	10101
−10	11010	10101	10110
−9	11001	10110	10111
−8	11000	10111	11000
−7	10111	11000	11001
−6	10110	11001	11010
−5	10101	11010	11011
−4	10100	11011	11100
−3	10011	11100	11101
−2	10010	11101	11110
−1	10001	11110	11111
−0	10000	11111	—
+0	00000	00000	00000
+1	00001	00001	00001
+2	00010	00010	00010
+3	00011	00011	00011
+4	00100	00100	00100
+5	00101	00101	00101
+6	00110	00110	00110
+7	00111	00111	00111
+8	01000	01000	01000
+9	01001	01001	01001
+10	01010	01010	01010
+11	01011	01011	01011
+12	01100	01100	01100
+13	01101	01101	01101
+14	01110	01110	01110
+15	01111	01111	01111

Considera-se, então, que B^n é o valor completo de um conjunto de números com *n* algarismos e, por isso, o complemento de cada um é o resultado da subtração de N por esse valor (é o que falta a N para completar o valor B^n). A Fig. 7.10 mostra alguns exemplos de complemento à base. Todos os exemplos consideram que os números a serem obtidos do complemento são números com cinco algarismos e, por isso, n = 5 (para simplicidade, não representamos os valores originais com 5 algarismos, mas com 2 algarismos em cada número, pois os que faltam para completar 5 são zeros e estão implicitamente colocados à esquerda do algarismo mais significativo de cada número).

Na base 10:	N = 763_{10}	C10 de N = $10^5 - N = 100\ 000_{10} - 763_{10} = 99\ 237_{10}$
Na base 8:	N = 254_8	C8 de N = $8^5 - N = 100\ 000_8 - 254_8 = 77\ 524_8$
Na base 2:	N = 110_2	C2 de N = $2^5 - N = 100\ 000_2 - 110 = 1\ 1\ 0\ 1\ 0_2$

Figura 7.10 Exemplos de valores representados em complemento à base.

Na prática, a obtenção mais rápida do valor do complemento à base de um número é realizada através da seguinte operação, em duas etapas:

a) Subtrair cada algarismo do maior algarismo da base considerada (9 na base 10, 1 na base 2 etc.).

b) Ao resultado encontrado somar 1 ao algarismo menos significativo (o mais à direita).

Os complementos dos números utilizados como exemplos na Fig. 7.10 podem ser calculados por esse método, assim:

$N = 763_{10}$	$C10 \text{ de } N = 99\,999 - 763 = 99\,236_{10} + 1 = 99\,237_{10}$
$N = 254_8$	$C8 \text{ de } N = 77\,777 - 254 = 77\,523_8 + 1 = 77\,254_8$
$N = 110$	$C2 \text{ de } N = 11\,111 - 110\ 11001_2 + 1\ 11010_2$

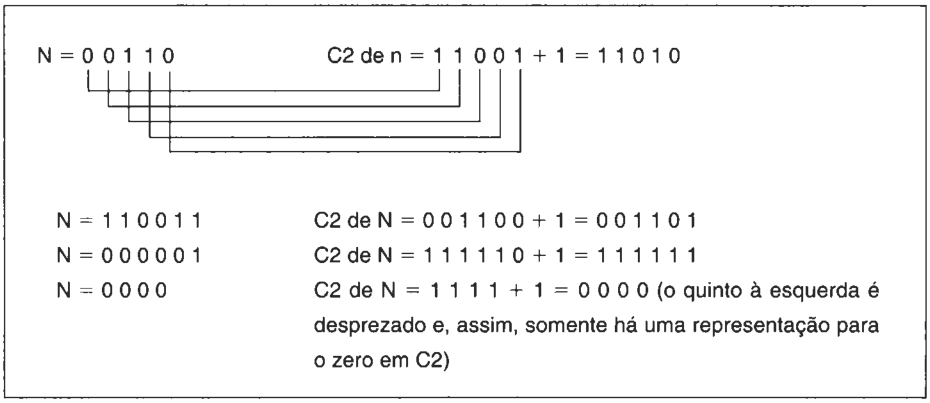
A primeira etapa desse método consiste na operação de obtenção do complemento à base −1, descrito no item Complemento à Base menos 1.

Quando se trata de valores na base 2 (números binários), pode-se executar a primeira etapa da obtenção do complemento a 2 de um número através de um outro método ainda mais rápido que o anterior:

- em vez de se subtrair cada algarismo do número do maior algarismo da base (no caso da base 2, trata-se de subtrair de 1), simplesmente inverte-se o valor de cada algarismo, isto é, se o algarismo é 0, passa a ser 1, e se for 1, passa a ser 0. A segunda etapa (somar 1 ao resultado) permanece a mesma.

A Fig. 7.11 mostra alguns exemplos desse método (todos os números estão representados na base 2).

Como o que interessa nesse texto é o trabalho interno nos computadores e esses são elementos binários, detalhes da representação e de operações aritméticas em complemento a 2 são mais importantes que complemento à base em geral.



N = 0 0 1 1 0   C2 de n = 1 1 0 0 1 + 1 = 1 1 0 1 0

N = 1 1 0 0 1 1   C2 de N = 0 0 1 1 0 0 + 1 = 0 0 1 1 0 1

N = 0 0 0 0 0 1   C2 de N = 1 1 1 1 1 0 + 1 = 1 1 1 1 1 1

N = 0 0 0 0   C2 de N = 1 1 1 1 + 1 = 0 0 0 0 (o quinto à esquerda é desprezado e, assim, somente há uma representação para o zero em C2)

Figura 7.11 Método rápido de obtenção de complemento de um número.

Pode-se dizer, sem risco de erro, que a quase totalidade dos computadores modernos utilizam aritmética de complemento a 2 (quando se trata de representação em ponto fixo), devido às duas grandes vantagens daquele método sobre sinal e magnitude, e mesmo sobre complemento a 1 conforme apresentado na Tabela 7.1:

- possuir uma única representação para o zero;
- necessitar de apenas um circuito somador para realizar, não só operações de soma, mas também operações de subtração (mais barato).

Conforme já pudemos observar no último exemplo da Fig. 7.11, a obtenção do complemento a 2 do valor 0 redunda no mesmo valor 0 sem troca de sinal; não há, portanto, duas representações para o zero, como acontece na representação em sinal e magnitude e complemento a 1.

Seja, por exemplo, um computador com palavra de 6 bits, isto é, com registradores e UAL com capacidade para armazenar números de 6 bits. Nesse caso, representa-se o valor zero como:

000000   o primeiro 0 indica o sinal do número.

Para calcular o complemento a 2 desse número, vamos usar o método rápido (da Fig. 7.11), ou seja: troca-se o valor de todos os bits, inclusive o bit de sinal, e adiciona-se 1 ao resultado. Assim, teremos:

$000000 \rightarrow 111111 + 1 = 000000$

O "vai 1" para a 7.ª ordem (à esquerda) é desprezado, porque consideramos o limite de 6 bits para o registrador. O valor final é igual ao inicial — uma única representação para o 0.

Isso acarreta uma assimetria na quantidade de números que podem ser representados em complemento a 2, permitindo que se tenha a representação de um número negativo a mais do que os números positivos. No caso exemplificado (com um registrador de 6 bits), teremos:

2^6 valores representáveis = 64 números binários.

Desses 64 números, 32 iniciam com 0 e 32 iniciam por 1. Os 32 números, cujo primeiro bit à esquerda é 0, representam 31 valores positivos (de +1 a +31) e o zero (000000), enquanto os 32 números com o primeiro bit à esquerda igual a 1 representam 32 valores negativos (−1 a −32).

Este número negativo a mais em relação aos números positivos decorre do fato de não haver representação de zero negativo; conseqüentemente este valor não tem um correspondente (com a mesma quantidade limite de bits da palavra, é claro) positivo. No exemplo anterior, o número −32 não tem correspondente positivo (+32).

Pode-se generalizar para qualquer quantidade de bits nos registradores, definindo a faixa de representação de números em complemento a 2 (ver Tabela 7.2):

$$\boxed{-2^{n-1} \quad \text{a} \quad +(2^{n-1} - 1)} \qquad (7.2)$$

Utilizando-se, por exemplo, um registrador de 6 bits, teremos n = 6 e n − 1 = 5; a faixa de representação será:

$$\boxed{\begin{array}{lll} -2^{6-1=5} & \text{a} & +(2^{6-1=5} - 1) \\ \text{de } -32 & \text{a} & +31 \end{array}}$$

(+32 não possui representação para essa quantidade de bits).

É importante, neste ponto, mencionarmos que complementar um número positivo significa torná-lo negativo, como por exemplo passar +13 para o valor −13. Isto é o mesmo que dizer: complementar +13, pois acarretará no valor −13. Da mesma forma, podemos voltar ao valor positivo, a partir de −13, complementando do mesmo modo o valor −13, o que redundará no valor +13.

De um modo geral, então, complementar N significa obter −N.

Esta observação é importante para entendermos a aritmética em complemento, bem como a representação de números naquela forma (de complemento).

Aritmética com complemento

Na aritmética (e na representação de números) com complemento, o sentido de negatividade do número não é dado por um símbolo que indica o sinal e é acrescentado ao número, mas que não é incluído na operação aritmética propriamente dita, como na representação em sinal e magnitude (símbolo + ou 0 e − ou 1). Em complemento a 2 (ou mesmo a 1), a negatividade é incorporada ao próprio número e, nesse caso, nas operações de soma e subtração, é usado o número completo (sem se dispensar o algarismo do sinal).

O ponto fundamental (diferença entre a aritmética de S&M e de C2) é que o complemento de um número N também representa a sua forma com o sinal inverso. Ou seja:

complemento de N = −N e complemento de − N = N

Em conseqüência, podemos estabelecer as seguintes conclusões:

1) A operação N1 + N2 é efetuada como uma soma comum, algarismo por algarismo (inclusive o algarismo de sinal para a aritmética binária). Os números negativos devem estar previamente representados em complemento à base.

2) A operação N1 − N2 = N1 + compl. à base de N2 ou NI + $\overline{\text{N2}}$
   - o traço horizontal sobre N2 significa que se trata da representação do complemento de N2;
   - a operação de soma se realiza de modo igual ao especificado no item 1.

Assim, ambas as operações, de adição e de subtração, são realizadas através do processo de soma, bastando pois um único componente somador na UAL, senão vejamos:

Operação soma: A + B = A + B (não interessa o sinal de A ou de B)

Operação de subtração: A − B = A + (−B) ou A + compl. de B ou ainda A + $\overline{B}$

Podemos demonstrar (e para isso utilizaremos como exemplos a aritmética decimal — mais bem entendida por nós) que operações aritméticas de soma e subtração podem ser efetuadas apenas empregando-se somas entre os números, desde, é claro, que esses estejam representados em complemento e que se utilize a aritmética de complemento.

Para simplificar, consideremos números decimais expressos com quatro algarismos para:

Exemplo 7.30

Somar (+15) e (−12). Os números em C10 terão quatro algarismos.

Solução

Efetuamos primeiramente a soma algébrica dos valores de modo a percebermos a exatidão dos resultados.

- soma algébrica: (+15) + (−12) = (+15) − (+12) = +3

Vamos agora executar a operação utilizando aritmética de complemento, sendo no caso complemento a 10 (complemento à base 10, pois os números estão representados na base 10).

- em C10: +15 = 0015 e −12 = 9988 (em ambos os casos desaparece o sinal)

−12 = 9988 porque teremos (9999 − 0012) + 1 = 9987 + 1 = 9988 (ver o item complemento à base)

Somando em C10, efetua-se a operação algarismo por algarismo. Operações aritméticas em complemento não usam sinal, o qual está incorporado ao número (algarismo mais significativo — mais à esquerda), e este algarismo também faz parte da operação como se o número estivesse sem sinal. Se ocorrer "vai 1" na soma dos dois algarismos mais significativos, ele será desprezado.

1 ←——— "Vai 1" para fora do número é desprezado, sem alterar o valor final correto.
0015
9988
0003 ←——— Resultado = +3 (porque temos um algarismo zero mais à esquerda do número)

Exemplo 7.31

Subtrair (+12) de (+19). Os números em C10 terão quatro algarismos.

- subtração algébrica: (+19) − (+12) = (+19) + (−12) = +7

O que fizemos foi transformar a operação de subtração em uma operação equivalente de soma, passamos +12 para −12 (complementamos +12 encontrando o valor oposto em sinal). Se trocamos o sinal do subtraendo, teremos que trocar o sinal da operação, que passará de subtração para soma. Deste modo o resultado se mantém igual e a verdadeira operação a ser realizada será de soma e não de subtração.

- N1 − N2 = N1 + compl. de N2 Como N2 = +12, então compl. de N2 = −12
- em C10: +19 = 0019 e −12 = 9988 (como no exercício anterior)

1 ←——— "Vai 1" desprezado.
0019
9988
0007 ←——— Resultado = +7 (por causa do algarismo 0 mais à esquerda)

Exemplo 7.32

Somar (−15) e (−12). Os números em C10 terão quatro algarismos.

- soma algébrica: (−15) + (−12) = −15 − 12 = (−27)
- em C10: −15 = 9999 − 15 = 9984 + 1 = 9985 e −12 = 9988 (como no exercício anterior)

```
1←——————  "Vai 1" desprezado.
 9985
 9988
 ————
 9973←——— Resultado, representado em C10 (pois tem um algarismo 9 na posição mais à esquerda)
```

Para sabermos qual é este valor em decimal teremos que complementar este resultado, ou seja:

9999 − 9973 = 22 + 1 = −23

O resultado é, então, −23.

Aritmética em Complemento a 2

A aritmética em complemento a 2 requer apenas um componente (somador) para somar dois números e um componente que realize a operação de complementação. O algoritmo básico refere-se, então, à soma dos números, considerando-se que os números negativos estejam representados em complemento a 2; ele acusa, também, se o resultado ultrapassar a quantidade de bits representáveis na UAL (e registradores), *overflow*, conforme já mostrado anteriormente (ver subitem 7.5.2).

Algoritmo da operação de adição em complemento a 2

a) Somar os dois números, bit a bit, inclusive o bit de sinal.

b) Desprezar o último "vai 1" (para fora do número), se houver.

c) Se, simultaneamente, ocorrer "vai 1" para o bit de sinal e "vai 1" para fora do número, ou se ambos não ocorrerem, o resultado está correto.

d) Se ocorrer apenas um dos dois "vai 1" (ou para o bit de sinal ou para fora), o resultado está incorreto. Ocorreu um *overflow*. O *overflow* somente pode ocorrer se ambos os números tiverem o mesmo sinal (seja positivo ou ambos negativos) e, nesse caso, se o sinal do resultado for oposto ao dos números.

Algoritmo da operação de subtração em complemento a 2

a) Complementar a 2 o subtraendo, independentemente se é um valor positivo ou negativo.

b) Somar ambos os números, utilizando o algoritmo da adição já mostrado antes.

Vamos considerar as quatro possibilidades indicadas nos algoritmos, de modo a confirmarmos sua correção. Elas são:

1) Ocorrência de "vai 1" para o bit de sinal e para fora do resultado.

2) Não ocorrência de nenhum "vai 1", nem para o bit de sinal nem para fora do número.

3) Ocorrência de "vai 1" para o bit de sinal mas não ocorre "vai 1" para fora do número.

4) Não ocorrência de "vai 1" para o bit de sinal mas ocorre "vai 1" para fora do número.

Nos exemplos a seguir (7.33 a 7.36) todos os números já estão representados em C2. Cada exemplo corresponde a uma das possibilidades relacionadas anteriormente.

Exemplo 7.33

Primeira possibilidade: ocorre "vai 1" para o bit de sinal e "vai 1" para fora do número resultante.

Somar 1100 e 1101.

Solução

Os números são negativos e estão representados em C2. Para sabermos seu valor decimal precisamos convertê-los para a representação em sinal e magnitude (S/M). Neste caso, teremos que trocar o valor dos bits da magnitude e somar 1 ao resultado, ou seja:

1011 + 1 = 1100 valor em S/M (por pura coincidência, o número apresenta os mesmos bits)

Como o bit mais significativo é 1, o número decimal é negativo.

A magnitude é 100 em binário, o que, em decimal, significa valor 4. O número 1100 é equivalente a decimal − 4.

De modo idêntico converte-se 1101 para S/M e tem-se: 1101 em C2 = 1011 em S/M. E 1011 em S/M = decimal − 3.

11 ←——	"Vai 1" para fora do número é desprezado.
1100	−4
+ 1101	−3
1001 ←——	O resultado está correto porque houve "vai 1" para o bit de sinal e "vai 1" para fora do número.

O resultado, 1001, é um valor negativo representado em C2. Passamos para S/M, o que redunda em 1111 (mantido o bit de sinal, trocam-se os valores dos bits da magnitude e soma-se 1 ao resultado). Este valor, 1111 em S/M representa, em decimal, ao valor: −7, o que garante a correção do resultado, pois (−3) + (−4) = (−7).

Exemplo 7.34

Segunda possibilidade: não ocorre "vai 1" para o bit de sinal nem ocorre "vai 1" para fora do número resultante.

Somar 0001 e 0101.

Solução

Os números são positivos e, portanto, sua representação é de S/M. Seu valor em decimal será:

0001 = +1 e 0101 = +5

←——	Não há nenhum "vai 1".
0001	+1
+ 0101	+5
0110 ←——	O resultado está correto porque não houve "vai 1" para o bit de sinal nem "vai 1" para fora do número.

O resultado, 0110 é um valor positivo e, neste caso, seu valor decimal é obtido da simples conversão de sua magnitude de base 2 para base 10, o que redunda no decimal +6. Este valor garante a correção do resultado, pois (+1) + (+5) = (+6).

Exemplo 7.35

Terceira possibilidade: ocorre "vai 1" para o bit de sinal, mas não ocorre "vai 1" para fora do número resultante.

Somar 0101 e 0110.

Solução

Os números são positivos e, portanto, sua representação é de S/M. Seu valor em decimal será:

0101 = +5 e 0110 = +6

$$\begin{array}{r l} 1\longleftarrow & \text{Há “vai 1” para o bit de sinal, mas não há “vai 1” para fora do número.} \\ 0101 & +5 \\ +\ 0110 & +6 \\ \hline 1011\longleftarrow & \text{O resultado está \textbf{incorreto} porque houve “vai 1”para o bit de sinal, mas} \\ & \text{não para fora do número.} \end{array}$$

Também podemos verificar a incorreção do resultado ao constatar que dois valores positivos somados não podem redundar em um valor negativo (bit de sinal — mais à esquerda é 1).

Esta incorreção decorre do excesso ao limite de representação de números, utilizando-se apenas 4 bits. Neste caso, a faixa permitida é de (−7 a +7), pois sendo n = 4 (quantidade de algarismos) teremos (ver expressão 7.2):

-2^{n-1} até $+ (2^{n-1} - 1)$, ou seja, $-2^{4-1} = -8$ até $+2^{4-1} - 1 = +7$

Ora, como o resultado de (+5) + (+6) = (+11) e a faixa limite de positivos é +7, o resultado deveria ser mesmo incorreto. Para que o resultado desse certo seria necessário que a palavra fosse igual a, pelo menos, 5 bits, cuja faixa de representação passaria a ser em decimal de −16 até +15.

Exemplo 7.36

Quarta possibilidade: não ocorre “vai 1” para o bit de sinal, mas ocorre “vai 1” para fora do número resultante.

Somar 1010 e 1101.

Solução

Os números são negativos e estão representados em C2. Para sabermos seu valor decimal precisamos convertê-los para a representação em sinal e magnitude (S/M). Neste caso, teremos que trocar o valor dos bits da magnitude e somar 1 ao resultado, ou seja:

1101 + 1 = 1110 valor em S/M

Como o bit mais significativo é 1, o número decimal é negativo.

A magnitude é 110 em binário, o que, em decimal, significa valor 6. O número 1010 é equivalente a decimal −6.

De modo idêntico converte-se 1101 para S/M e tem-se: 1101 em C2 = 1011 em S/M. E 1011 em S/M = decimal −3.

$$\begin{array}{r l} & \text{Não há “vai 1” para o bit de sinal, mas há “vai 1” para fora do número.} \\ 1\longleftarrow & \text{Este “vai 1” é desprezado.} \\ 1010 & -6 \\ +\ 1101 & -3 \\ \hline 0111\longleftarrow & \text{O resultado está \textbf{incorreto} porque não houve “vai 1” para o bit de sinal, mas} \\ & \text{houve para fora do número.} \end{array}$$

Também podemos verificar a incorreção do resultado ao constatar que dois valores negativos somados não podem redundar em um valor positivo (bit de sinal — mais à esquerda é 0).

Esta incorreção decorre do excesso ao limite de representação de números, utilizando-se apenas 4 bits. Neste caso, a faixa permitida é de (−7 a +7), pois sendo n = 4 (quantidade de algarismos) teremos (expressão 7.2):

-2^{n-1} até $+(2^{n-1} - 1)$, ou seja, $-2^{4-1} = -8$ até $+2^{4-1} - 1 = +7$

Ora, como o resultado de (−6) + (−3) = (−9) e a faixa limite de negativos é −8, o resultado deveria ser mesmo incorreto. Para que o resultado desse certo seria necessário que a palavra fosse igual a, pelo menos, 5 bits, cuja faixa de representação passaria a ser em decimal de −16 até +15.

Observe-se um fato interessante: quando se trata de realizar uma operação aritmética e esta é de subtração, complementa-se o subtraendo, isto é, se o valor é A passa a ser −A, seja A um valor positivo (por exemplo, +6, que se tornaria −6) ou um valor negativo (por exemplo, −6, que passaria a ser +6). Em binário, todos os bits do número, inclusive o de sinal, serão invertidos e soma-se 1 ao resultado.

No entanto, quando o resultado de uma operação é um valor negativo, representado naturalmente em C2 e deseja-se saber seu valor decimal, precisa-se converter o valor de C2 para S/M e, assim, trocam-se também todos os bits da magnitude e soma-se 1 ao resultado. Neste caso, como se viu, trocam-se os bits apenas da magnitude mantendo-se o bit 1 de sinal invariável. Isto é claro, visto que não se está complementando o número e sim apenas obtendo outra representação (de C2 para S/M) do mesmo número negativo. Se trocássemos o valor do bit de sinal, o número passaria a ser positivo.

Vamos consolidar o entendimento dessa aritmética simples em complemento a 2 através de novos exemplos, tendo sempre em mente que:

a) As operações de soma são normalmente realizadas como soma.

b) As operações de subtração são realizadas como soma de complemento (minuendo mais o complemento do subtraendo).

c) Se o resultado encontrado é um valor positivo, então o valor decimal correspondente da magnitude é obtido por pura conversão de base 2 para base 10.

d) Se o resultado encontrado é um valor negativo, deve-se primeiro converter esse valor para representação de sinal e magnitude (consistirá em trocar o valor dos bits da magnitude e somar 1 ao resultado) e, em seguida, converter a magnitude de base 2 para base 10.

Exemplo 7.37

Efetuar a adição dos seguintes números: (+13) e (+15). Considerar palavra de 6 bits.

Solução

+13 = 0 01101 e +15 = 0 01111

Efetuando a operação algébrica, temos: (+13) + (+15) = (+28). Como a faixa de representação de valores em C2, com palavra de 6 bits, é de −32 até +31 (já vimos isso nos exercícios anteriores e expressão 7.2), a operação deverá indicar um resultado correto.

001111 ← Não houve "vai 1" para o bit de sinal nem para fora do número.

001101

+ 001111

011100 ← Resultado correto, porque não houve "vai 1" nem para bit de sinal nem para fora do número.

Resultado positivo (bit de sinal = 0) e magnitude = 11100 = decimal 28. Ou seja: +28.

Exemplo 7.38

Efetuar a adição dos seguintes números: (+23) e (+20). Considerar palavra de 6 bits.

Solução

+23 = 0 10111 e +20 = 0 10100

Efetuando a operação algébrica, temos: (+23) + (+20) = (+43). Como a faixa de representação de valores em C2, com palavra de 6 bits, é de −32 até +31 (expressão 7.2), a operação deverá indicar um resultado incorreto (*overflow*) visto que +43 é maior que o limite positivo (+31) com 6 bits de palavra.

```
 011100 ←—— Houve “vai 1” para o bit de sinal, mas não houve para fora do número.
 010111
+ 010100
 111001 ←—— Resultado incorreto porque houve “vai 1” para bit de
            sinal e não para fora do número.
```

Resultado incorreto: *overflow.*

Exemplo 7.39

Efetuar a adição dos seguintes números: (+15) e (−13). Considerar palavra de 6 bits.

Solução

+15 = 0 01111 e −13 em S/M = 1 01101 e em C2 = 1 10010 + 1 = 1 10011

Efetuando a operação algébrica, temos: (+15) + (−13) = (+2). Como a faixa de representação de valores em C2, com palavra de 6 bits, é de −32 até +31 (expressão 7.2), a operação deverá indicar um resultado correto.

A operação é realizada com a adição em binário de +5 com −13 representado em C2, pois o algoritmo para a aritmética em C2 somente é válido se os valores negativos estiverem previamente representados em C2, independentemente do fato de que venha o número a ser complementado ou não.

```
 111111 ←—— Houve “vai 1” para o bit de sinal e para fora do número; este é desprezado.
 001111
+ 110011
 000010 ←—— Resultado correto porque houve “vai 1” para bit de sinal e para fora do número.
```

Resultado positivo (bit de sinal = 0) e magnitude = 00010 = decimal. Ou seja: +2.

Exemplo 7.40

Efetuar a subtração: (+20) − (+17). Considerar palavra de 6 bits.

Solução

+20 = 0 10100 e +17 = 0 10001

Efetuando a operação algébrica, temos: (+20) − (+17) = (+20) + (−17) = +3. Observamos, então, que foi usado o conceito de substituição da operação de subtração por soma de complemento. −17 é o complemento de +17. Como a faixa de representação de valores em C2, com palavra de 6 bits, é de −32 até +31 (expressão 7.2), a operação deverá indicar um resultado correto, pois o resultado será +3.

Para realizar a operação complementa-se o valor 010001 (+17) e, nesse caso, trocam-se todos os bits (inclusive o de sinal) e soma-se 1 ao resultado, obtendo-se o equiv1Tente, em binário, ao número −17. Este complemento fica assim:

C2 de 010001 = 101110 + 1 = 101111

```
 111100 ←—— Houve “vai 1” para o bit de sinal e para fora do número; este é desprezado.
 010100
+ 101111
 000011 ←—— Resultado correto porque houve “vai 1” para bit de sinal e
            para fora do número.
```

Resultado positivo (bit de sinal = 0) e magnitude = 00011 = decimal 3. Ou seja: +3.

Exemplo 7.41

Efetuar a subtração: (−24) − (−15). Considerar palavra de 6 bits.

Solução

−24 em S/M = 1 11000 e em C2 = 1 00111 + 1 = 1 01000 −15 em S/M = 1 01111 e em C2 = 1 10000 + 1 = 1 10001

Efetuando a operação algébrica, temos: (−24) − (−15) = (−24) + (+15) = −9. Observamos, então, que foi usado o conceito de substituição da operação de subtração por soma de complemento, sendo +15 o complemento de −15. Como a faixa de representação de valores em C2, com palavra de 6 bits, é de −32 até +31 (expressão 7.2), a operação deverá indicar um resultado correto, pois o resultado será −9.

Para realizar a operação complementa-se o valor 110001 (−15) e, nesse caso, trocam-se todos os bits (inclusive o de sinal) e soma-se 1 ao resultado, obtendo-se o equivalente, em binário, ao número +15. Este complemento fica assim:

C2 de 110001 = 001110 + 1 = 001111

```
  001000←——— Não houve "vai 1" para o bit de sinal nem para fora do número.
  101000
+ 001111
  ------
  110111←——— Resultado correto porque não houve "vai 1" para bit de sinal nem
             para fora do número.
```

Resultado negativo (bit de sinal = 1) e como está representado em C2, para conhecermos seu valor decimal, teremos que converter para S/M. Assim:

1 10111 em C2 = 1 01000 + 1 = 1 01001 em S/M (trocam-se os bits da magnitude e soma-se 1 ao resultado.

Sendo o valor em S/M = 101001, então em decimal o número é −9, o bit de sinal é 1 ("−") e a magnitude 1001 = 9.

Na realidade, um algoritmo mais "inteligente" analisaria os sinais dos números e efetuaria diretamente a soma dos valores positivos, em vez de complementá-los duas vezes. Porém, se o valor (−15) estivesse armazenado em complemento a 2 (110001), antes de a soma ser iniciada ele apenas seria complementado a 2 (001111), que representa (+15).

Exemplo 7.42

Seja A = 10111100 e B = 00110011 (ambos usando palavra de 8 bits e já representados em C2).

Efetuar a operação: A − B, usando aritmética de C2 (isto significa: usar o algoritmo para operação aritmética em C2).

Solução

Estando os valores já representados em C2, temos que B = +51, pois bit de sinal é 0 e magnitude 0110011 = 51.

Como A é negativo (bit de sinal = 1) deve ser convertido para S/M para sabermos seu valor decimal.

10111100 em C2 = 11000011 + 1 = 11000100 em S/M = −68

Utilizando-se o algoritmo para aritmética em C2 temos que: A − B = A + C2 de B. Então:

A = 10111100 e B = 00110011, sendo C2 de B = 11001100 + 1 = 11001101

```
   11111100←———— Houve "vai 1" para bit de sinal e para fora do número; este é desprezado.
   10111100
+  11001101
 ——————————
   10001001←———— Resultado correto, pois ocorreram ambos os "vai 1".
```

O resultado é um número negativo (bit de sinal igual a 1) e representado em C2. Portanto, terá que ser convertido para S/M para conhecer-se seu valor decimal.

Sendo em 10001001 em C2, será 11110110 + 1 = 11110111 em S/M.

11110111 em S/M = −119

Como A = −68 e B = +51, temos que: A − B = (−68) − (+51) = −68 − 51 = −119 (resultado correto, pois está dentro da faixa de representação de números em C2 com palavra de 8 bits (a faixa é de −128 até +127).

Exemplo 7.43

Seja A = 00110001 e B = 10000011 (ambos usando palavra de 8 bits e já representados em C2).

Efetuar a operação: −A + B, usando aritmética de C2 (isto significa: usar o algoritmo para operação aritmética em C2).

Solução

Estando os valores já representados em C2, temos que A = +49, pois bit de sinal é 0 e magnitude 0110001 = 49.

Como B é negativo (bit de sinal = 1) deve ser convertido para S/M para sabermos seu valor decimal.

10000011 em C2 = 11111100 + 1 = 11111101 em S/M = −125

Utilizando-se o algoritmo para aritmética em C2 temos que: −A + B = C2 de A + B. Para obter-se o C2 de A trocam-se todos os bits (inclusive o de sinal) e soma-se 1 ao resultado:

A = 00110001 e C2 de A = 11001110 + 1 = 11001111

```
   10001111←———— Não houve "vai 1" para bit de sinal e houve para fora do número;
   11001111      este é desprezado.
+  10000011
 ——————————
   01010010←———— Resultado incorreto, pois ocorreu apenas "vai 1" para fora do número.
```

Observa-se que o resultado é mesmo incorreto, pois dois valores negativos somados não poderão resultar em um valor positivo. Pela expressão 7.2 observa-se que a faixa de representação com palavra de 8 bits é de −128 até +127. No presente exemplo temos que:

A = +49 e B = −125. E que −A + B = −(+49) + (−125) = −49 − 125 = −174, que excede −128 e, por isso, temos o *overflow*, resultando no valor incorreto.

Complemento à Base Menos Um

A definição matemática do complemento à base menos um (C_{B-1}) de um número N é:

$C_{B-1}(N) = (2^n - N) - 1$, onde n = quantidade de algarismos do número e $2^n - N$ = complemento à base ou C_B

Como se pode observar, também é possível dizer que se trata da representação em complemento à base subtraída de 1 (um); daí o nome complemento à base menos um.

Na prática, obtém-se o complemento à base menos um de um número (C_{B-1}), subtraindo-se do valor (B − 1) cada algarismo do número; o valor B − 1 é o maior algarismo da base, seja ela qual for.

Por exemplo, se a base B é 10, então C_{B-1} = complemento a 9 e obtém-se, de forma prática, o complemento a 9 de um número decimal, subtraindo-se do valor 9 o algarismo correspondente do número, um a um. Assim, o complemento a 9 do número 716 é o número 283, porque:

$9 - 7 = 2$

$9 - 1 = 8$

$9 - 6 = 3$ ou 283

Se a base for 2, teremos complemento a 1 (2 − 1), que será obtido subtraindo-se de 1 (B − 1) cada algarismo do número. Na realidade, pode-se obter a representação em complemento a 1 de um número através da simples troca de valor dos bits do número; ou seja: altera-se o valor de cada bit, de 1 para 0 e de 0 para 1.

Exemplos de complemento a 1:

C1 de 01110 → 10001

C1 de 11001 → 00110

C1 de 00000 → 11111

Nessa forma de representação (válida apenas para valores negativos), também temos duas representações para o zero. E, assim, a faixa limite de representação de números em complemento a 1 é idêntica à de sinal e magnitude.

$$\boxed{-(2^{n-1} - 1) \quad \text{a} \quad +(2^{n-1} - 1)} \qquad (7.3)$$

A Tabela 7.2 exemplifica o formato das três representações em ponto fixo (sinal e magnitude, complemento a 1 e complemento a 2), utilizando-se valores com 5 bits.

Aritmética em complemento à base menos 1

Como já mencionado, os computadores vêm sendo fabricados com UAL que realiza apenas aritmética em complemento a 2 (para representação em ponto fixo). Isso é devido, basicamente, à maior rapidez e simplicidade do método C2, bem como à única representação para o zero (CI e S&M possuem dupla representação para o zero).

Por essa razão, vamos nos deter, neste item, apenas no essencial. Somente mencionaremos elementos básicos de C1, de modo a não detalhar informações sobre uma representação já em desuso.

O algoritmo para operações de soma em C1 é ligeiramente diferente, embora guarde os mesmos princípios básicos de C2. Naturalmente, como em C2, a operação de subtração é realizada através da soma do minuendo com o complemento a 1 do subtraendo, ou seja:

A − B = A + C1 de B

a) Somar os dois números bit a bit, inclusive o bit de sinal.

b) Contar a ocorrência de "vai 1", para o bit de sinal e para fora do número (se ocorrer algum).

c) O "vai 1" para fora do número (se houver) será somado ao resultado obtido. Nessa soma, continuam-se contando os "vai 1" para o bit de sinal e para fora do número que ocorrer.

d) O resultado estará correto se a quantidade de "vai 1" contada for par. Caso contrário, o resultado estará incorreto, pois ocorreu *overflow*.

Exemplo 7.44

Adicionar (+15) e (+13). Considerar palavra de 6 bits.

Solução

+15 = 0 01111 e +13 = 0 01101

001111 ← Quantidade de "vai 1" é zero (nem bit de sinal nem para fora).
001111
+ 001101
011100 ← Resultado correto, pois a quantidade de "vai 1" é zero.

O valor em decimal do resultado é: +28 (bit de sinal 0 − "+" e magnitude 11100 = 28). Este valor está dentro da faixa limite de representação de números em C1 com palavra de 6 bits (expressão 7.3), que é de −31 até +31. Razão por que também o resultado está correto.

Exemplo 7.45

Adicionar (+25) e (+13). Considerar palavra de 6 bits.

Solução

+25 = 0 11001 e +13 = 0 01101

011001 ← Quantidade de "vai 1" é 1 — ÍMPAR (1 para bit de sinal e 0 para fora do número).
011001
+ 001101
100110 ← Resultado incorreto, pois a quantidade de "vai 1" é ímpar.

Verifica-se que o resultado é incorreto devido a três fatos:

1) A quantidade de "vai 1" é ímpar (apenas o "vai 1" para o bit de sinal).
2) Dois valores positivos somados não podem resultar em um valor negativo.
3) O resultado esperado em decimal excede o limte da representação com este tamanho de palavra, 6 bits (expressão 7.3), que é de −31 até +31, pois (+25) + (+13) = +38.

Exemplo 7.46

Adicionar (−18) e (−7). Considerar palavra de 6 bits.

Solução

(−18) em S/M = 1 10010 e em C1 = 1 01101 (trocar os bits da magnitude) e (−7) em S/M = 1 00111 e em C1 = 1 11000

→ 111000 ← Quantidade de "vai 1" é PAR; 1 para bit de sinal e 1 para fora do número; este é adicionado ao resultado.
101101
+ 111000
100101
→ + 1
100110 ← Resultado correto; quantidade de "vai 1" é igual a 2, PAR.

O valor em decimal do resultado é um número negativo e como está representado em C1 necessita ser convertido para S/M para identificar-se seu valor decimal. Temos, pois:

100110 em C1 = 111001 em S/M = −25 (bit de sinal 1, "−" e magnitude 11001 = 25).

Este valor está dentro da faixa limite de representação de números em C1 com palavra de 6 bits (expressão 7.3), que é de −31 até +31. Razão por que também o resultado está correto.

Exemplo 7.47

Adicionar (−17) e (−14)

Solução

(−17) em S/M = 1 10001 e em C1 = 1 01110 (trocar os bits da magnitude) e (−14) em S/M = 1 01110 e em C1 = 1 10001

$$\begin{array}{r} \rightarrow 100000 \\ 101110 \\ +\ \ 110001 \\ \hline 011111 \\ \rightarrow +\ \ 1 \\ \hline 100000 \end{array} \longleftarrow$$

Resultado correto; a quantidade de "vai 1" é igual a 2, PAR, sendo 1 "vai 1" para fora do número, na primeira soma e 1 "vai 1" para bit de sinal na segunda soma.

O valor em decimal do resultado é um número negativo e como está representado em C1 necessita ser convertido para S/M para se identificar seu valor decimal. Temos, pois:

100000 em C1 = 111111 em S/M = −31 (bit de sinal 1, "−" e magnitude 11111 = 31).

Este valor está dentro da faixa limite de representação de números em C1 com palavra de 6 bits (expressão 7.3), que é de −31 até +31. Razão por que também o resultado está correto.

7.5.2 Overflow

Quando uma soma de dois números de n algarismos resulta em um valor com $n + 1$ algarismos, ocorre o que chamamos *overflow*.

Esse é um fato válido, sejam os números binários ou decimais, com ou sem sinal.

Quando realizamos tal operação com lápis e papel, não sentimos esse problema porque o papel é sempre largo o suficiente para colocarmos mais um algarismo à esquerda, no resultado.

No entanto, em computadores com circuitos digitais, em que todos os seus elementos são finitos, a quantidade de algarismos permitida para representar um valor é sempre finita e limitada e, assim, *overflow* se torna um problema porque todas as memórias, principalmente os registradores, têm tamanho fixo e finito. Não há lugar para um bit extra (vai 1 para fora do número).

Por isso, todo sistema de computação verifica, de algum modo, a ocorrência de *overflow* em operações aritméticas (uma das maneiras foi mostrada nos subitens anteriores). Sua ocorrência é sinalizada para que o sistema operacional tome alguma providência a respeito (como, por exemplo, ao utilizar um bit de um registrador especial, que assume o valor 1 quando ocorre *overflow*, o sistema sempre verifica o valor desse bit após uma operação aritmética).

Se a aritmética for realizada em sinal e magnitude (como fazemos com lápis e papel), a ocorrência de *overflow* é facilmente detectada pelo bit "vai 1" após o último bit de magnitude à esquerda.

Porém, na soma por complemento, os algoritmos que foram descritos acarretam a modificação completa do resultado se ocorrer *overflow*. Por exemplo, dois números positivos somados, resultando em um valor de resultado negativo, ou vice-versa. Isso ocorre porque a soma em complemento inclui também a soma dos bits de sinal; o "vai 1" para o bit de sinal, somado a eles, modifica seu valor, alterando a natureza do número.

Na representação e aritmética com números reais, fracionários, pode ocorrer não somente *overflow* como também *underflow*. Este último se caracteriza por ocorrer um resultado cujo valor é menor que o menor valor representável com uma específica quantidade de bits estabelecida para a forma de representação em uso. Esta forma, para números reais, é em geral a de ponto flutuante, que será apresentada no item a seguir.

7.5.3 Representação em Ponto Flutuante

Em muitos cálculos de Engenharia, Física, Astronomia, Matemática etc., os dados são números de valor muito grande, como a massa da Terra, a distância entre a Terra e um astro qualquer do universo; outras vezes, os números considerados são demasiadamente pequenos, tais como a massa de um átomo, ou de um elétron.

Se esses números tivessem que ser representados em ponto fixo (qualquer que seja o método: S&M, C1 ou C2), seria necessário utilizar uma grande quantidade de algarismos, muito mais do que a UAL de qualquer computador pode normalmente armazenar em ponto fixo. E pior ainda: a grande maioria dos algarismos seria de valor igual a zero.

Vejamos um exemplo:

Se N1 = 0,0000000000000000000000000073 e N2 = 25370000000000000000000000000

e se fosse desejado efetuar a operação de soma:

S = N1 + N2

teríamos que efetuar assim:

25370000000000000000000000000,0000000000000000000000000000

+ 000000000000000000000000000000,0000000000000000000000000073

O que consumiria 56 algarismos decimais (imaginem quantos mais seriam necessários se a representação fosse em números binários).

Uma das possíveis soluções para esse problema seria criar algoritmos para eliminar os zeros na ocasião de imprimir o valor; porém, internamente, seria um consumo caro e desnecessário de memória e UCP para realizar tais operações com todos esses algarismos.

O método mais simples e eficiente empregado para resolver o problema consiste na utilização da representação conhecida na Matemática como *notação científica* e que, em computação, é denominada *ponto flutuante (floating point).* Nessa forma de representação, pode-se representar números muito grandes ou números muito pequenos sem consumir enormes quantidades de algarismos, visto que só se usam as quantidades necessárias de algarismos, além dos valores serem representados por potências e, por isso, não requererem muitos dígitos. Por exemplo, o número +5 bilhões representado em forma inteira de ponto fixo necessita de 11 símbolos: +5.000.000.000. No entanto, em notação científica precisaríamos apenas de 7 símbolos:

$+5 \times 10^{+9}$ (e se considerarmos que o valor da base, 10, é sempre o mesmo, precisaríamos apenas de 5 símbolos)

Um número em notação científica é representado por um produto de dois fatores: o primeiro fator indica o sinal do número mais a sua parte significativa, sua precisão, e a segunda parte indica a grandeza do número, representada por uma potência; é justamente o valor do expoente que marca essa grandeza. Assim, temos:

$N = \pm F \times B^{\pm E}$, onde:

N = número que se deseja representar;
$\pm$ = sinal do número;
F = dígitos significativos do número (também chamada parte fracionária ou mantissa);
B = base de exponenciação;
$\pm E$ = valor do expoente, com seu sinal (o expoente pode ser positivo, valores acima de 1, ou negativo, valores compreendidos entre 0 e 1).

Observação: A representação do expoente é definida pelo fabricante do processador. Pode-se usar representações do tipo *sinal e magnitude, complemento a 2* e *excesso de N.* A representação sinal e magnitude é a mais simples de descrever e exemplificar, razão por que será utilizada nos exemplos e exercícios. A representação usando-se complemento a 2 é implementada da mesma forma descrita anteriormente, sendo usada em alguns sistemas devido à rapidez da execução das operações de soma e subtração dos expoentes.

Como não há necessidade de explicar mais ainda as formas sinal e magnitude e complemento a 2, exaustivamente já mostradas, resta apenas uma breve descrição da forma excesso de N.

Basicamente, no método excesso de N o valor do expoente em geral é obtido como se descreve a seguir.

Efetua-se a seguinte operação de subtração: $(2^n/2) + E = C$, onde:

n é a quantidade de algarismos do campo reservado para o expoente,

E é o valor real do expoente e

C (alguns fabricantes chamam de característica) é o valor a ser armazenado no campo do expoente.

A utilização do processo de representar números por um produto permite uma separação entre a precisão desejada para o valor (algarismos significativos, expressos na fração ou mantissa) e a grandeza do número (expressa pelo expoente).

Desse modo, aqueles valores dos números N1 e N2 do exemplo inicial podem ser representados em notação científica como

$N1 = +0{,}73 \times 10^{-24}$ e $N2 = 0{,}2573 \times 10^{+30}$

Nesse caso, somente os algarismos significativos de cada número foram usados (73 para N1 e 2573 para N2); todos os zeros foram substituídos por poucos algarismos representativos do valor do expoente.

Um outro fato pode ser observado: tendo em vista que a grandeza do número é expressa pelo valor do expoente, os números N1 e N2 podem ser representados por diferentes produtos, todos com o mesmo valor final:

$N1 = 0{,}73 \times 10^{-24} = 0{,}0073 \times 10^{-22} = 73 \times 10^{-26}$

$N2 = 2573 \times 10^{+26} = 25{,}73 \times 10^{+28} = 0{,}2573 \times 10^{+30}$

O conceito é o mesmo qualquer que seja tanto a base de numeração utilizada quanto a base de exponenciação, que têm significado diferente da base de representação.

Em computadores, onde qualquer valor é representado internamente em base 2, teríamos, por exemplo, um dado número:

$X = 1010011{,}1011$ ou $X = 0{,}\ 10100111011 \times 2^{+111\,(+7)}$

Internamente, o sistema precisa armazenar:

a) O sinal do número: bit 0 para positivos e bit 1 para negativos.

b) O valor da mantissa: 10100111011(pode-se esquecer o 0, por ser desnecessário, visto que todos os números iniciam por "0," estes símbolos podem deixar de ser representados internamente, poupando bits, mas o sistema "sabe" que o número inicia por "0,").

c) O valor do expoente e seu sinal: +7 = 0 111 (o 0 indica sinal "+", como seria 1 se o sinal fosse "−").

Outro exemplo:

$N4 = 111000111001$ ou $N4 = 0,\ E\ 39 \times 16^{+3}$

Nesse caso, foi usada a base de exponenciação 16 apenas para verificarmos que essa base pode ser diferente da base 10 ou da base 2.

A base de exponenciação não precisa ser armazenada para cada número (como o sinal, a mantissa e o expoente precisam), pois o sistema reconhece sua existência e valor por ser igual para qualquer número.

Desse modo, em uma representação de números em ponto flutuante, dois fatores são basicamente considerados:

- precisão do número — expressa pela quantidade de algarismos da parte fracionária ou mantissa
- grandeza do número — expressa pelo valor do expoente.

A precisão mede a exatidão do número, como, por exemplo:

(Pi) pode ser expresso como: 3,14 ou 3,1416 ou 3,141592 etc.

Cada representação mais à direita tem maior precisão que a da esquerda adjacente, pois é expressa com mais algarismos significativos.

A grandeza, ou limite de representação, indica quão grande ou quão pequeno é o número que se deseja representar. O valor do campo expoente de uma representação em ponto flutuante indica, então, a faixa de grandeza dos números naquela representação.

Por exemplo, se em uma certa representação em ponto flutuante o campo expoente possuir 6 bits de tamanho (serão 5 bits para o valor do expoente e 1 bit para seu sinal), isso significa que teremos disponíveis para representar números tão pequenos quanto 2^{-31} e tão grandes quanto 2^{+31}. Isto é, uma faixa de representação com 2^{62} valores.

7.5.3.1 Representação Normalizada

Conforme pudemos verificar pelos exemplos precedentes, a representação de um número em ponto flutuante (notação científica) pode ser efetuada através de diferentes produtos (diferentes valores de expoente para diferentes posições da vírgula fracionária na mantissa).

Por exemplo, o número $+25_{10}$ pode ser representado, em notação científica, como:

$+25 = +25 \times 101^{0}$

$+25 = +0{,}25 \times 10^{+2}$

$+25 = +0{,}0025 \times 10^{+4}$

$+25 = +2500 \times 10^{-2}$, e assim por diante.

Para evitar diferentes interpretações de representação, costuma-se estabelecer nos sistemas uma representação-padrão denominada representação *normalizada.*

Na forma normalizada, a parte fracionária ou mantissa é definida sempre como sendo um valor M (ou F) que satisfaça a seguinte expressão:

$1/B \leq M < 1$ (exceto se $M = 0$)

Na prática, essa definição pode ser expressa pelas seguintes regras:

a) A mantissa deve ser sempre fracionária (M é sempre menor que 1).

b) O primeiro algarismo após a vírgula tem que ser diferente de zero (porque é igual ou maior que 1/B).

Se a base de exponenciação (base B) for 2, então o primeiro algarismo após o 0 será sempre 1 (a mantissa poderá ser qualquer número entre 1/2 e 1 (excluído 1).

Os seguintes valores estão normalizados:

0,23510 0,11012

1.° algarismo diferente de 0.

Os seguintes valores NÃO estão normalizados:

0,023510 0,0011012

1.° algarismo é 0.

A forma normalizada apresenta uma grande vantagem, que é permitir a melhor precisão possível, por conter apenas algarismos significativos.

7.5.3.2 Conversão de Números para Ponto Flutuante

Para um dado sistema, a representação em ponto flutuante é especificada a partir da identificação dos seguintes elementos:

a) A quantidade de palavras de dados (total de bits/bytes da representação).

b) O modo de representação da parte fracionária ou mantissa (se normalizada, se em C2, se em S&M etc.).

c) O modo de representação do expoente (S&M, C2, por característica etc.).

d) A quantidade de bits definida para o expoente e para a mantissa.

e) A posição; no formato, do sinal do número, da mantissa e do expoente.

f) O valor da base de exponenciação.

A título de exemplo, vamos definir um formato típico de representação em ponto flutuante. A Fig. 7.12 mostra o formato da representação, com o tamanho em bits de cada campo.

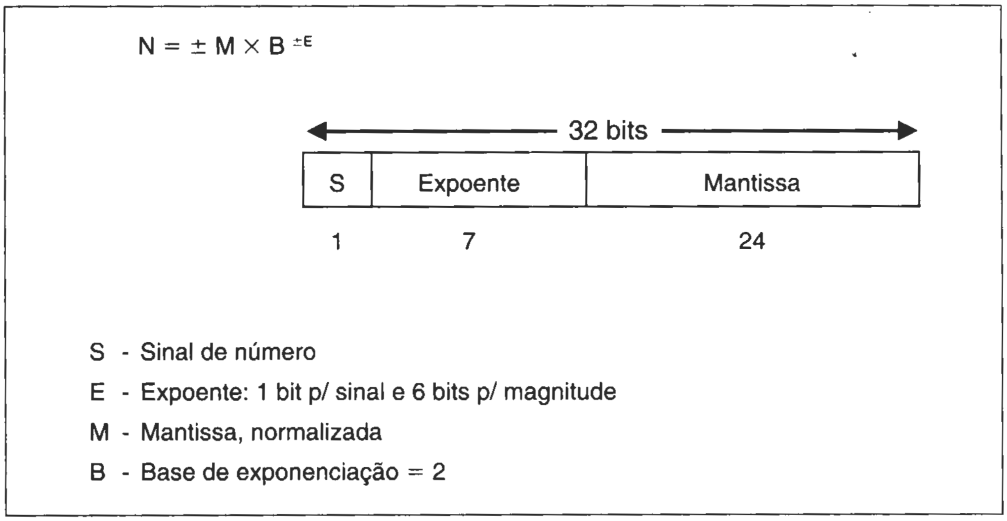

N = ± M × B ±E
32 bits
S
Expoente
Mantissa
1
7
24
S - Sinal de número
E - Expoente: 1 bit p/ sinal e 6 bits p/ magnitude
M - Mantissa, normalizada
B - Base de exponenciação = 2

Figura 7.12 Um formato de representação em ponto flutuante.

Exemplo 7.48

Converter o valor decimal +407,375 para ponto flutuante (formato da Fig. 7.12).

Solução

O que se deseja é obter valores binários para o sinal do número (S), para a parte significativa ou mantissa (M) e para o expoente (E), de modo a satisfazer a expressão:

$N = \pm M$ (ou F) $\times B^{\pm E}$

onde:

$N = 407{,}375$

Sinal do número = +(positivo)

M e E — ainda desconhecidos

Deste modo, temos:

$407{,}375 = +M \times 2^{\pm E}$, sendo M fracionária (1.° bit não zero).

A primeira etapa do processo consiste na conversão direta do valor decimal para seu correspondente valor em algarismos binários, ou seja: conversão de um número da base 10 para a base 2 (item 3.3.3), o que vai identificar todos os algarismos (bits) significativos do número, ou seja, sua parte fracionária ou mantissa:

$407{,}375_{10} = 110010111{,}011_2$

Esse valor já poderia ser assumido como o valor da mantissa (algarismos significativos) e, nesse caso, para satisfazer a representação em notação científica (produto de dois fatores), o expoente seria igual a zero:

$407{,}375_{10} = 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1,\ 0\ 1\ 1 \times 2^0$

Porém, como a forma da mantissa deve ser normalizada (a mantissa deve ser apenas uma fração, não tendo parte inteira à esquerda da vírgula e tendo o primeiro algarismo após a vírgula diferente de zero), é necessário deslocar a vírgula fracionária da posição atual para a posição situada imediatamente à esquerda do algarismo mais significativo (2.ª etapa).

O número passaria de 110010111,011 para: 0, 110010111 011

deslocamento da vírgula

Como o deslocamento da vírgula (nove casas para a esquerda) corresponde à divisão do número 9 vezes por 2 (divisão por 2^9) — visto que a operação é em base 2 — para manter íntegro. o valor do número é necessário multiplicá-lo por 2^9. Isso é obtido através do incremento do valor do expoente (incrementar nove vezes), que passa do valor 0 para o valor 9 (3.ª etapa). A expressão, em notação científica (ponto flutuante), ficaria assim:

$407{,}375 = 0{,}1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1 \times 2^{+9\ (1001)}$

Já podemos, então (4.ª etapa), indicar todos os valores dos campos constantes do formato de representação em ponto flutuante (Fig. 7.12):

S (sinal do número) = 0 (número positivo)

E (expoente) = 0 0 0 1 0 0 1 (+9 → 0 = sinal positivo e 001001 = 9, magnitude)

M (fração ou mantissa) = 110010111011000000000000 (dispensa-se indicar o valor 0,). A mantissa tem 24 algarismos (Fig. 7.12) e, portanto, completamos seu valor com zeros à direita, por se tratar de um valor fracionário.

A representação completa do número em ponto flutuante seria em binário (como fica internamente no sistema) e em hexadecimal (para simplificar a apresentação), de acordo com o formato dos campos indicado na Fig. 7.12:

Binário	00001001110010111011000000000000
Hexadecimal	0 9 C B B 0 0 0

Exemplo 7.49

Converter o valor decimal −0,078125 para representação em ponto flutuante, de acordo com o formato especificado na Fig. 7.12.

Solução

Seguindo as etapas realizadas de acordo com o Exemplo 7.48, teremos:

1.ª etapa — conversão da magnitude do número de base 10 para base 2:

$0{,}078125_{10} = 0{,}000101_2$

2.ª etapa — transformação do número para o formato de ponto flutuante (notação científica):

$-0{,}078125 = -0{,}000101 \times 2^0$ (a mantissa já é fracionária)

3.ª etapa — normalização da mantissa:

deslocamento da vírgula três ordens para a direita e ajustar o expoente

$-0{,}078125 = -0{,}101 \times 2^{-3}$ (mantissa × 3 e expoente/3)

4.ª etapa — indicação dos valores de cada campo do formato (Fig. 7.12):

S = 1 (número negativo)

E = 1000011 (−3) (bit mais significativo, mais à esquerda, de sinal = 1 e magnitude 000011 = 3)

M = 101000000000000000000000 (completa-se o valor 101 com zeros à direita até 24 bits)

Formato final em binário e em hexadecimal:

Binário	11000011101000000000000000000000
Hexadecimál	C 3 A 0 0 0 0 0

Exemplo 7.50

Converter o valor armazenado na memória de um computador (mostrado a seguir em hexadecimal) em formato de ponto flutuante (definição apresentada na Fig. 7.12) para seu correspondente valor decimal.

O valor hexadecimal armazenado é: 0 4 D 0 0 0 0 0

Solução

As etapas a serem seguidas para obtenção do resultado final, valor em decimal do número, são:

1) Converter o valor de hexadecimal para binário

   04D00000 = 00000100110100000000000000000000

2) Separar os 32 bits encontrados de acordo com o formato especificado na Fig. 7.12, ou seja:

   S − E − M, sendo: S = 1 bit; E = 7 bits, 1 bit de sinal mais 6 bits da magnitude e M = 24 bits

Assim, temos:

0	0	000100	110100000000000000000000
S_N	S_E	E	M

3) Identificar cada campo com os bits correspondentes, a seguir mostrado:

   S_N = 0. O sinal do número é bit 0, sendo, portanto, positivo ("+").

   S_E = 0. O sinal do expoente é bit 0, sendo, portanto, positivo ("+").

   E = 000100. Convertendo de binário para decimal, temos $100_2 = 4_{10}$.

   M = 110100000000000000000000 ou como valor fracionário: 0,110100000000000000000000

4) Mostrar o número completo na forma de notação científica:

   N = +0,11010000000000000000 × 2^{+4} ou simplificando (suprimindo os zeros) N = 0,1101 × 2^{+4}

5) Calcular a expressão indicada pelo produto anterior

   N = 1101. Pois multiplicando 0,1101 quatro vezes por 2 significa andar com a vírgula 4 ordens para a direita.

   $1101_2 = 13_{10}$

Para confirmar o resultado final podemos fazer de outra maneira, convertendo o valor binário da mantissa, 0,1101, para seu valor decimal correspondente, que é: 0,8125. Em seguida, multiplicar este valor por 16, que corresponde a 2^4.

0,8125 × 16 = 13

Assim, o resultado final é $N = 13_{10}$.

A forma de representação mostrada na Fig. 7.12 não é a única existente. Sistemas de computação comerciais possuem outras formas semelhantes, mas com certas diferenças (por exemplo, alguns fabricantes empregam a representação em complemento a 2 para o tipo de dado do campo expoente). Um dos aspectos mais interessantes dessas diferenças consiste na diversidade de precisão definida por fabricantes, conforme o processador seja mais dedicado a fins comerciais ou científicos, estes últimos requerendo sempre uma precisão maior em vários tipos de cálculos.

A Tabela 7.3 apresenta características de alguns sistemas antigos (computadores de grande e médio porte), bem como o formato padrão definido pelo IEEE — Institute of Electrical and Electronics Engineers Inc. — padrão IEEE 754, 1985. Os atuais processadores, sejam os Pentium da Intel, sejam os da AMD, Cyrix ou Motorola/Apple, seguem o padrão IEEE 754.

Aritmética em ponto flutuante

Como os números são representados em ponto flutuante pela expressão

$N = \pm M \times B^{\pm E}$

as operações aritméticas devem ser realizadas considerando aquele produto; as mantissas e os expoentes devem ser manipulados separadamente.

Tabela 7.3 Formatos de Representações em Ponto Flutuante

Sistema	Total de bits	Base de exponenciação	Quantidade bits do expoente	Quantidade de bits da mantissa
Grande porte IBM — formato reduzido	32	16	7	24
Grande porte IBM — formato estendido	128	16	15	112
Médio porte DEC — VAX — precisão simples	32	2	8	23
Grande porte CYBER — precisão simples	120	2	11	108
IEEE 754 — precisão simples	32	2	8	23
IEEE 754 — precisão dupla	64	2	11	52
IEEE 754 — precisão estendida	80	2	15	64

Adição e subtração

São operações mais complexas em ponto flutuante do que em aritmética de ponto fixo, devido à necessidade de alinhamento da vírgula (ponto) fracionária.

A operação de adição ou subtração é efetivamente realizada pela soma ou subtração dos valores das mantissas das parcelas, desde que o valor dos expoentes de cada uma seja igual. A igualdade dos expoentes indica que o alinhamento da vírgula fracionária está correto.

Assim, considerando os números N1 e N2, expressos em notação científica:

$N1 = M1 \times B^{\pm E1} \qquad N2 = M2 \times B^{\pm E2}$

Obteremos os valores de soma (N_S) e da diferença (N_D), sendo M_S e M_D, respectivamente, o resultado da soma e o da diferença entre os valores das mantissas, e E_S e E_D, respectivamente, o valor do expoente resultante na soma e na diferença:

$Ns = N1 + N2 = Ms \times B^{\pm Es} \qquad Nd = N1 - N2 = Md \times B^{\pm Es}$

Sendo que:

Ms = M1 + M2 e Md = M1 − M2

Desde que:

Es = E1 = E2 e Ed = E1 = E2

A operação de soma ou subtração das mantissas é uma simples operação em ponto fixo (em sinal/magnitude ou complemento a 2, conforme o sistema de computação usado); o resultado da operação deve ser normalizado, caso já não esteja.

Antes da operação aritmética com os valores das mantissas, deve ser verificada a igualdade dos expoentes; caso seus valores sejam diferentes, deve ser efetuada a operação de ajuste da igualdade dos valores.

O processo de ajuste da igualdade dos expoentes é, em geral, realizado de modo que o expoente resultante seja o maior dos dois valores. Ou seja, o sistema efetua a subtração entre os valores dos expoentes. Em seguida, a mantissa de menor valor é dividida (deslocamento à direita — "*shift right*") pelo valor da diferença. Finalmente, o expoente do número menor é igualado ao do maior, que passa a ser o expoente do resultado.

Exemplo 7.51

Efetuar adição dos números N1 = +37 e N2 = −9, utilizando o formato de representação em ponto flutuante da Fig. 7.12.

Solução

Para melhor compreensão do leitor, o exemplo será solucionado realizando-se as tarefas em partes, como a seguir detalhado:

1) Conversão de N1 e N2 para a representação em ponto flutuante (na prática, esses valores já estariam representados em ponto flutuante, tarefa realizada pelo compilador (ver Cap. 9) tendo em vista a definição da variável, efetuada pelo programador no seu programa-fonte).

   $+37 = 0{,}100101 \times 2^{+6} = 00000110100101000000000000000000$
   $-9 = 0{,}1001 \times 2^{+4} = 10000100100100000000000000000000$

2) A soma é realizada somente se os expoentes tiverem o mesmo valor e, neste caso, somam-se as mantissas. Como os expoentes não são de mesmo valor, é necessário antes ajustar seus valores. Isto é realizado do seguinte modo:

   Subtraindo o menor do maior expoente:

   $6 - 4 = 2$

   Como E2 é o menor dos expoentes e a diferença entre seus valores é 2, deve-se somar 2 ao menor expoente (E2 + 2 = 4 + 2 = 6) e assim ele se iguala a E1. Para N2 se manter de mesmo valor, deve-se dividir sua mantissa, M2, por 2^2, assim:

   N2 é multiplicado por 2^2 (pela soma de 2 ao seu expoente) e ao mesmo tempo é dividido por 2^2 (pela divisão de sua mantissa por 2^2).

   $M2/2^2 = 0{,}1001/2^2 = 0{,}001001$

   E2 = +6, que é também o valor de E1 e, portanto, do resultado E_S

3) Somar algebricamente as mantissas. No caso, a soma algébrica resulta em M1 − M2 (porque M1 é "+" e M2 é "−"):

   $Ms = 0{,}100101 - 0{,}001001 = 0{,}0111$ (não se registram os zeros à direita)

4) Normalizar a mantissa do resultado:

   Passa de Ms = 0,0111 para Ms = 0,111 (deslocamento da vírgula uma vez para a direita — $Ms \times 2^1$)

   Passa de Es para 6 − 1 = 5 (corresponde à divisão do número por 2^1).

5) O formato completo do valor do resultado, Ns, é:

   S = + = 0 (sinal do maior dos números)

   E = 0 000101 (+5)

   M = 11100000000000000000000

   Então:

Ns = 0	0000101	11100000000000000000000
SE	M	
+	2^{+5}	0,111

   ou seja: $Ns = +0{,}111 \times 2^{+5} = 11100 = +28_{10}$

   Verificando, temos: (+37) + (−9) = +28

   Finalmente, em hexadecimal:

   $\boxed{Ns = M1 + M2 = 0\,5\,E\,0\,0\,0\,0\,0}$

Multiplicação e divisão

As operações de multiplicação e divisão com números representados em ponto flutuante também requerem a manipulação separada da mantissa e do expoente, não havendo, porém, necessidade da operação de alinhamento da vírgula (que é uma operação demorada), que somente diz respeito a operações de adição e subtração.

Em essência, dados os números N1 e N2:

$N1 = \pm M1 \times B^{\pm E1}$ e $N2 = \pm M2 \times B^{\pm E2}$

Temos o seu produto Np = N1 × N2 e a sua divisão Nd = N1/N2, representados da seguinte forma:

$Np = \pm Mp \times B^{\pm Ep}$ e $Nd = \pm Md \times B^{\pm Ed}$

Sendo que a obtenção dos campos S_P, E_P e M_P, componentes do produto N_P e dos campos S_D, E_D e M_D, componentes do resultado da divisão, N_D e cujos formatos estão especificados na Fig. 7.12, se realiza da seguinte forma:

1) Sinal do produto e da divisão

   Se N1 e N2 tiverem o mesmo sinal, então o sinal de Np ou de N_D será positivo ("+"), bit 0.

   Se N1 e N2 tiverem sinais diferentes, então o sinal de Np ou de N_D será negativo ("−"), bit 1.

2) Expoente do produto ou da divisão:

   E = E1 + E2 (soma algébrica)

   E = E1 − E2 (subtração algébrica)

3) Mantissa do produto e da divisão:

   Mp = M1 × M2

   Md = M1/M2

Exemplo 7.52

Multiplicar os números N1 = +89 e N2 = −19, utilizando as regras definidas para a aritmética com números representados em ponto flutuante e adotando o formato da Fig. 7.12.

Solução

1) Conversão de N1 e N2 para formato de ponto flutuante (item 7.5.3.2):

   $+89 = +0{,}1011001 \times 2^{+7} = 00000111101100100000000000000000$

   $-19 = -0{,}10011 \times 2^{+5} = 10000101100110000000000000000000$

2) Determinação do sinal do produto (Sp):

   Como S1 $\neq$ S2, então: Sp = −(corresponde ao bit 1)

3) Cálculo da mantissa do produto (Mp):

   MP = M1 × M2 = 0,1011001 × 0,10011 = 0,011010011011 (não registrando os zeros)

   A multiplicação é realizada na UAL conforme o tipo de dado adotado (complemento a 2 etc.).

   Observe que a mantissa não está normalizada, pois o 1.° algarismo após a vírgula é zero.

4) Cálculo do expoente do produto (Ep):

   $Ep = E1 + E2 = (+7) + (+5) = +12_{10}$

5) Normalização do resultado

   Para que o 1.° algarismo após a vírgula (na mantissa) seja diferente de zero, desloca-se a vírgula uma vez para a direita (corresponde a multiplicar o valor uma vez pela base — no caso, é a base 2 —, ou seja, multiplicar por 2^1). Mantém-se o mesmo valor final dividindo-se o número pelo mesmo valor: 2^1. Para se obter isso, efetua-se a subtração de 1 do expoente Ep (que sendo atualmente 12 passa a ser 11).

   Mp passa de 0,011010011011 para 0,11010011011

   $Ep = Ep - 1 = +12 - 1 = +11_{10} = 0001011_2$

6) Formato final do produto Np, na representação de ponto flutuante:

   10001011110100110110000000000000 ou

   8 13 13 3 6 0 0 0 (em hexadecimal)

   Para conferir se o resultado está correto efetua-se a multiplicação em base decimal e também realiza-se o cálculo do produto indicado pelo formato de ponto flutuante (notação científica):

   $+89_{10} \times -19_{10} = -1691_{10}$ e

1	0	001011	110100110110000000000000
S_N	S_E	E	M

   $N = 0,11010011011 \times 2^{+11} = 11010011011_2 = 1024 + 512 + 128 + 16 + 8 + 2 + 1 = 1691_{10}$

Exemplo 7.53

Efetuar a divisão de $N1 = +264_{10}$ por $N2 = +0,75_{10}$, utilizando aritmética de ponto flutuante e o formato apresentado na Fig. 7.12.

Solução

1) Conversão de N1 e N2 para o formato de ponto flutuante:

   $+264 = +0,100001 \times 2^{+9}$ = 00001001100001000000000000000000

   $+0,75 = +0,11 \times 2^{+0}$ = 00000000110000000000000000000000

2) Determinação do sinal do resultado (S_D):

   Como S1 = S2, então: S_D = +(corresponde ao bit 0)

3) Cálculo do valor da mantissa do resultado (M_D):

   $M_D = M1/M2 = 0,100001/0,11 = 0,1011$ (não se registrando os zeros à direita)

   A divisão é realizada pela UAL, segundo o algoritmo correspondente ao tipo de dado adotado (complemento a 2 etc.)

4) Cálculo do expoente do resultado (Ed):

   E = E1 − E2 = (+9) − (+0) = +9 = 0 001001

5) Normalização da mantissa do resultado

   Não é necessário porque o primeiro algarismo após a vírgula, na mantissa, é diferente de zero.

6) Formato final do resultado da divisão, N, na representação em ponto flutuante:

   00001001101100000000000000000000

   0 9 B 0 0 0 0 0

Para conferir se o resultado está efetivamente correto efetua-se a divisão em base decimal e depois de acordo com o produto indicado pela notação científica:

$+264_{10}/0,75_{10} = 352_{10}$

$+0,1011_2 \times 2^{+9} = 101100000_2 = 256 + 64 + 32 = 352_{10}$

7.5.4 Representação Decimal

As formas de representação de dados numéricos descritas nos itens anteriores (ponto fixo e ponto flutuante) são eficientes e adequadas para utilização em cálculos matemáticos, de engenharia e em outras áreas de ciências matemáticas ou afins.

No entanto, possuem certas desvantagens em aplicações comerciais, especialmente no caso de representação e operações matemáticas com valores financeiros, devido à necessidade de resultados absolutamente exatos em nível de centavos (ou cents ou pennies etc.) e não apenas aproximadamente precisos (a representação de um número em ponto flutuante pode mostrá-lo com dezenas de algarismos significativos, indicando grande precisão, mas ainda assim é um resultado aproximado, não exato).

O fechamento de um balanço de banco, por exemplo, deve ser obtido com toda a exatidão e não apenas com aproximação.

Para isso, é útil existir uma forma de representar e operar com valores na sua forma decimal, para que os valores obtidos nas operações matemáticas sejam decimais e, conseqüentemente, exatos.

Porém, não se cogita construir um sistema de computação com representação interna de valores em outro modo que não o binário, devido a fortes razões econômicas e técnicas.

A solução encontrada para equilibrar a necessidade de realizar eventuais operações aritméticas com valores decimais, mas representando-os internamente, sempre na forma binária, constitui-se num método híbrido de representação de dados (nem completamente decimal nem totalmente binária) denominado *código binário decimal* (Binary Coded Decimal ou BCD).

Na forma BCD (há outros métodos de representação decimal, porém menos conhecidos na prática), os dados decimais usados em um programa, em vez de serem diretamente convertidos da base 10 para a base 2, são representados internamente por códigos binários correspondentes a cada algarismo decimal, conforme mostrado na Tabela 7.4.

Assim, por exemplo, o número decimal 7458 seria representado no código BCD da seguinte forma:

0111	**0100**	**0101**	**1000**
7	**4**	**5**	**8**

Uma vez que com 4 bits podemos codificar 16 valores diferentes e, na base 10, somente possuímos dez algarismos, há realmente um desperdício de códigos, o que pode ser uma desvantagem dessa representação, quando se trata de codificar números grandes. Por exemplo, representar em BCD o número 1.734.345.200 requer o emprego de 40 bits (10 algarismos × 4 bits por algarismo) mais uma quantidade de bits para indicar o sinal do número. Este mesmo valor seria representado em ponto fixo com apenas 31 algarismos binários, 31 bits.

Entre os algarismos sem código válido em decimal (códigos representativos dos valores decimais de 10 a 15), é comum utilizar alguns deles para indicar o sinal do número. Há sistemas que adotam a seguinte convenção para o sinal dos números representados em BCD:

1100 → representa o sinal positivo ("+")

1101 → representa o sinal negativo ("−")

Na maioria dos sistemas de computação que possuem a forma de representação decimal, um dos métodos mais usados é o BCD, denominado *decimal compactado* (packed decimal) para diferençar de um método mais antigo e em desuso, o *decimal zonado* (zoned decimal).

Na representação BCD ou decimal compactado, o número decimal é convertido algarismo por algarismo ao código binário descrito na Tabela 7.4. O sinal é colocado após o algarismo menos significativo à direita, segundo uma das convenções já apresentadas.

Tabela 7.4 Representação de números no formato BCD

Decimal	Binário — BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Operações aritméticas em BCD

As operações aritméticas realizadas pela UAL de um sistema de computação com números representados na forma decimal BCD não têm a mesma simplicidade que o mesmo processo realizado com números representados em ponto fixo.

Um dos fatores que prejudicam a execução dos algoritmos para operações aritméticas em decimal é justamente o fato de os números não estarem representados na forma binária pura, mais bem entendida pelo hardware.

Outro detalhe refere-se aos números de valor decimal entre 10_{10} e 15_{10} (correspondem aos números binários de 1010 a 1111), que não representam algarismo válido na base 10. O algoritmo da soma em BCD deve prever essa situação, o que torna sua execução mais demorada.

Operação de adição

Na definição de um algoritmo para realizar o processo de adição de dois números, usando aritmética em BCD, deve ser considerado que:

a) cada algarismo decimal é constituído por um valor com 4 bits e, portanto, a soma entre dois algarismos decimais é, na realidade, efetuada entre quatro algarismos binários de cada vez;

b) o hardware deve poder somar as duas parcelas e mais o "vai 1" que foi gerado na adição dos algarismos anteriores;

c) o resultado da adição de uma parcela pode ultrapassar o valor do maior algarismo decimal (algarismo 9), podendo ser, nesse caso, igual a um dos valores existentes entre 10_{10} a 18_{10}; por isso deve ser definida uma regra que evite um resultado incorreto.

Na realidade, o resultado da soma de duas parcelas em aritmética BCD pode ser enquadrado em uma das três seguintes categorias:

- Valor igual ou menor que 9. Resultado correto e definitivo; não há passagem de "vai 1" para a soma da parcela seguinte.
- Valor maior que 9 e igual ou menor que 15. Soma-se ao resultado o valor decimal 6 (0110_2) de modo a produzir um resultado válido. Não há passagem de "vai 1" para a soma da parcela seguinte.

- Valor maior que 15 e igual ou menor que 18. Soma-se 6 ao resultado; há passagem de "vai 1" para a soma da parcela seguinte.

O valor 6 a ser somado ao resultado é devido, justamente, à existência de 6 números (10 a 15) não-válidos.

A seguir, é apresentado um algoritmo para a soma com números representados em BCD e alguns exemplos que auxiliam a compreensão de cada uma das possibilidades definidas para o resultado.

Algoritmo para a operação de soma em BCD — (A + B)

1) Decompor os números A e B em grupos de 4 bits, cada grupo representando um dos algarismos decimais dos números, de acordo com o código mostrado na Tabela 7.4.
2) Somar os novos números A e B (aritmética binária), como se fossem valores binários puros (bit a bit).
3) Se o resultado da soma dos quatro primeiros algarismos for igual ou menor que 1001, ele está correto; retornar ao item 1 para a adição da parcela seguinte de 4 bits.
4) Se o resultado for superior a 1001, soma-se 0110 (6_{10}) àquele valor; o novo resultado é o algarismo desejado; o "vai 1" após o último bit à esquerda é ainda acrescentado à soma da parcela seguinte. Retornar ao item 1.
5) Se o resultado for superior a 1111, somar 0110 ao resultado; o "vai 1" obtido com essa última soma é transferido para a parcela seguinte. Retornar ao item 1.
6) O processo acaba após a adição do último grupo de 4 bits (algarismo decimal mais significativo).

Exemplo 7.54

Adicionar A = $+3_{10}$ e B = $+5_{10}$. Utilizar a representação e o correspondente algoritmo de aritmética BCD.

Solução

Em BCD, conforme a Tabela 7.4, os valores decimais passam a ser: $3_{10} = 0011_2$ e $5_{10} = 0101_2$.

$$\begin{array}{r} 0011 \\ +\ 0101 \\ \hline 1000 \longleftarrow \end{array}$$

Resultado correto; não há "vai 1" após a soma do último bit de cada parcela.

O valor decimal do resultado é obtido da conversão do binário 1000 para decimal = 8. Também se verifica que está correto através da adição decimal das parcelas: 3 + 5 = 8.

Exemplo 7.55

Adicionar A = 5_{10} e B = 6_{10}. Utilizar a representação e o correspondente algoritmo de aritmética BCD.

Solução

Em BCD, conforme a Tabela 7.4, os valores decimais passam a ser: $5_{10} = 0101_2$ e $6_{10} = 0110_2$.

$$\begin{array}{r} 0101 \\ +\ 0110 \\ \hline \rightarrow 1 \longleftarrow \\ 1011 \\ +\ 0110 \\ \hline \rightarrow 1\ 0001 \\ \uparrow\ \ \uparrow \\ 1\ \ 1 = 11_{10} \end{array}$$

Como o resultado (1011) ultrapassa (1001 = 9), então é como se ocorresse "vai 1" na operação (na realidade, em decimal, o resultado de uma adição sendo 11, ocorre "vai 1".

O resultado está correto. Passa-se o "vai 1" para a parcela seguinte (próximo grupo de 4 bits, se houvesse, ou para o resultado, como neste caso).

Assim, o valor resultante, em decimal, será composto de 2 algarismos, o primeiro à direita (menos significativo) é igual a 1, correspondente ao valor binário encontrado: 0001. O outro algarismo (à esquerda) é também igual a 1, mas trata-se do "vai 1" encontrado na primeira soma, por ter excedido 1001 (9).

Exemplo 7.56

Adicionar A = 8_{10} e B = 9_{10}. Utilizar a representação e o correspondente algoritmo de aritmética BCD.

Solução

Em BCD, conforme a Tabela 7.4, os valores decimais passam a ser: $8_{10} = 1000_2$ e $9_{10} = 1001_2$.

```
1◄──────
 1000
+1001
──────
1 0001
+0110
──────
1 0111
▲  ▲
│  │
│  │
│  │
1  7 = 17₁₀
```

Este "vai 1" passa para a parcela seguinte da soma.

Resultado parcial (0001, corresponde ao algarismo decimal 1).

Soma-se 0110 = 6 ao resultado parcial, mantendo-se o "vai 1" anterior, que desce para o novo resultado.

Resultado final correto, com dois algarismos decimais: 17. O algarismo 7 corresponde ao valor binário 0111 (resultado da operação) e o algarismo 1 corresponde ao "vai 1" encontrado na operação inicial, pois o resultado excedeu 1001 (9).

Exemplo 7.57

Adicionar A = 1734_{10} e B = 4985_{10}. Utilizar a representação e o correspondente algoritmo de aritmética BCD.

Solução

Em BCD, conforme a Tabela 7.4, os valores decimais passam a ser: 1734_{10} = 0001 0111 0011 0100_2 e 4985_{10} = 0100 1001 1000 0101_2.

```
  0001 0111 0011 0100            1734
+ 0100 1001 1000 0101           +4985
─────────────────────           ─────
                                 6719

                 1001  (1)
         '1 1011       (2)
            0110
─────────────────────
     1 0000 0001 1001
  0101 0110
─────────────────────
  0110 0111 0001 1001
   ▲    ▲    ▲    ▲
   │    │    │    │
   6    7    1    9
```

(1) A soma do primeiro algarismo decimal de cada número (4 + 5) corresponde à soma do primeiro grupo de 4 bits (0100 + 0101), cujo resultado é 1001 = 9. Pelo algoritmo, o resultado está correto e passa-se para a parcela seguinte (próximo grupo de 4 bits).

(2) A soma destes, 0011 e 1000 = 1011 = 3 + 8 = 11, portanto, ultrapassa 1001 = 9. Neste caso, soma-se 0110 (6) ao 1011, e o "vai 1" encontrado deve ser transferido para a parcela seguinte.

Operação de subtração em aritmética BCD

A operação de subtração de números representados na forma decimal é pouco simples e prática, devido ao fato de não ser possível converter de modo rápido os números negativos para sua representação de complemento.

Conforme já exposto no item 7.5.1, o melhor método de efetuar operações de subtração consiste no emprego da aritmética de complemento (evitando-se a operação de subtração). No entanto, com os valores representados em BCD, a simplicidade da complementação a 2 de um número binário (substituição do bit 0 pelo bit 1, do bit 1 por 0 e somar 1 ao resultado) deixa de existir, visto que a complementação em BCD pode acarretar valores incorretos.

Por exemplo, o complemento a 2 do valor 0010 (decimal 2) é 1110_2 (decimal 14), o que, além de não ser um algarismo decimal válido, não é o natural complemento a 10 do valor 2 (o complemento a 10 de 2_{10} é 8_{10}).

Pode-se, então, identificar três possíveis maneiras para efetuar operações de subtração com números em formato BCD:

1) Realizar a efetiva operação de subtração na forma usual, com "empréstimos" e definição do sinal do resultado — mesmas regras de operação com sinal e magnitude.
2) Substituir o código de representação BCD por um outro código decimal, como, por exemplo, o de representação "excesso de 3" (ver Tabela 7.5) e efetuar a operação por complemento a 9.
3) Utilizar a representação e regras de complemento a 9 ou complemento a 10, com formato BCD.

Tabela 7.5 Representação de algarismos na forma "excesso de 3", com seus complementos a 9, C9 e a 10, C10

Algarismo decimal	Código em excesso de 3	Complemento a 9	Complemento a 10
0	0011	1100	1101
1	0100	1011	1100
2	0101	1010	1011
3	0110	1001	1010
4	0111	1000	1001
5	1000	0111	1000
6	1001	0110	0111
7	1010	0101	0110
8	1011	0100	0101
9	1100	0011	0100

Primeiro método — Subtração pelo método usual

É o processo menos prático, mais complexo e demorado, razão de seu atual desuso.

Exemplo 7.58

Seja A = 351_{10} e B = 237_{10}. Efetuar a operação A − B, utilizando representação BCD.

Solução

No formato BCD (ver Tabela 7.4), os valores são: 351 = 0011 0101 0001 e 237 = 0010 0011 0111.

A operação em si é realizada da seguinte forma:

$$\begin{array}{r} 0100\ 1010 \\ 0011\ 0101\ 0001 \\ \hline \underline{0001}\ \ \underline{0001}\ \ \underline{0100} \\ \uparrow \qquad \uparrow \qquad \uparrow \\ 1 \qquad 1 \qquad 4 \end{array} \qquad \begin{array}{r} 351 \\ \hline 114 \end{array}$$

Segundo método — Subtração utilizando o código "excesso de 3"

Conforme se pode verificar pela Tabela 7.5, o código "excesso de 3" é formado adicionando-se o valor 3 a cada algarismo decimal e, em seguida, convertendo-se o novo valor na correspondente representação binária usual.

Esse código (parece bem esquisito) tem a vantagem de permitir a obtenção direta do complemento a 9 ou complemento a 10 de qualquer número, diferentemente do método, nada prático, de se obter C9 e C10 para valores representados em BCD.

Por exemplo, a representação do algarismo 5_{10} é:

$5_{10} \rightarrow \% + 3 = 8_{10} \rightarrow 1000_2$ (excesso de 3)

O complemento a 9 é obtido fazendo-se, normalmente, o complemento a 1:

C9 de 8 → C1 de 1000 = 0111 (troca dos bits) → corresponde ao decimal 4.

Ora, o complemento a 9 de 5_{10} é o número 4_{10}.

O algoritmo para se efetuar a subtração por complemento de números representados no código "excesso de 3" consiste, basicamente, nas mesmas etapas para operações em complemento (C2 para C10, e C1 para C9) de ponto fixo, explicadas, respectivamente, nos itens 7.5.1.2 e 7.5.1.3. Há apenas algumas alterações decorrentes do processo de formação em "excesso de 3":

1) Inicialmente os valores devem estar representados no código "excesso de 3".
2) Os números negativos devem ser representados em C9 (ou em C10).
3) Somar os valores, tendo em vista que:
   A − B = A + compl. (B) → compl. (B) significa C9 ou C10 de B.
4) A soma é realizada bit a bit, como se os valores fossem números binários puros.
   4a) Se na soma de um grupo de 4 bits (código de um algarismo decimal) não ocorrer "vai 1" para fora do grupo, deve-se subtrair 3 (ou adicionar $13_{10}) = 1101_2$) ao resultado.
   4b) Se ocorrer "vai 1" para fora do grupo, deve-se adicionar $3_{10} = 0011_2$ ao resultado.
5) No caso de a operação ser realizada com valores em C9, se ocorrer "vai 1" após o último bit do último algarismo decimal (último grupo de 4 bits), deve-se adicionar 1 ao resultado geral.

Exemplo 7.59

Considerando-se os números A = 351 e B = 257, efetuar a operação A − B, utilizando-se a representação "excesso de 3".

Solução

No formato "excesso de 3" (ver Tabela 7.5), os valores são: 351 = 0110 1000 0100 e 237 = 0101 0110 1010.

O complemento a 10 de B, C10 (B) (ver Tabela 7.5) = 1010 1001 0110.

A operação será realizada assim:

$$\begin{array}{rcccl}
\text{desprezar} \rightarrow & 1 & 1 & 0 & \longleftarrow \text{“Vai 1” para o próximo algarismo decimal (grupo de 4 binários)} \\
 & 0110 & 1000 & 0100 & \\
 & 1010 & 1001 & 0110 & \\ \cline{2-4}
 & 0001 & 0001 & 1010 & \\
 & 0011 & 0011 & 1101 & \\ \cline{2-4}
 & \underline{0100} & \underline{0100} & \underline{0111} & \\
 & \uparrow & \uparrow & \uparrow & \\
 & 1 & 1 & 4_{10} & \text{(ver Tabela 7.5)}
\end{array}$$

Terceiro método — Subtração por complemento — representação em BCD

Esse método procura atingir a vantagem do uso da aritmética em complemento (subtração por soma de complemento), mas ainda possui a desvantagem da obtenção do C10 ou C9, onde não se realiza a simples troca de 0s por 1s e 1s por 0s.

Nesse caso, é necessário que haja, na UAL, um dispositivo específico para realizar à complementação de cada algarismo decimal (complementa um grupo de 4 bits).

O algoritmo se torna idêntico à operação de soma no formato BCD.

Exemplo 7.60

Efetuar a subtração de A = 351 e B = 257, A − B, utilizando-se a representação BCD, aritmética de complemento.

Solução

No formato BCD (ver Tabela 7.4), os valores são: 351 = 0011 0101 0001 e 237 = 0010 0011 0111.

A operação em si é realizada da seguinte forma:

$$\begin{array}{ccc}
0011 & 0101 & 0001 \\
0111 & 0110 & 0011 \\ \hline
1 & & \\
1010 & 1011 & 0100 \\
0110 & 0110 & \\ \hline
\underline{0001} & \underline{0001} & \underline{0100} \\
\uparrow & \uparrow & \uparrow \\
1 & 1 & 4_{10}
\end{array}$$

Multiplicação e divisão

As operações de multiplicação e divisão para números decimais são bem mais complexas que as mesmas operações com números binários. Isso porque, na multiplicação binária, o algarismo multiplicador só pode assumir os valores 0 e 1; o produto parcial, então, somente poderá ser ou zero (0) ou o próprio valor do multiplicando (quando o multiplicador é 1).

Já no caso da multiplicação com números decimais, os algarismos do multiplicador podem ter os valores entre 0 e 9 e, em conseqüência, a obtenção do produto parcial é mais complexa devido às possíveis variações de valores.

Na matemática (em computadores, é raro implementar-se esse tipo de operação), há diversas soluções para o problema, sendo a mais simples (e também a mais demorada) a de realizar a multiplicação por sucessivas somas. Ou seja, soma-se o multiplicador a ele próprio tantas vezes quanto o valor do multiplicado, exatamente como se realiza essa operação com lápis e papel.

Por motivos semelhantes, a operação de divisão com valores decimais é complexa e de execução demorada em face da quantidade de algarismos diferentes envolvidos, enquanto, na divisão binária, o quociente pode ter apenas os valores 0 e 1, e, na divisão decimal, os algarismos do quociente podem ter os valores de 0 a 9.

Um dos processos usados para realizar divisão decimal, semelhante ao da divisão binária, consiste em subtrair o divisor do dividendo sucessivamente enquanto o dividendo for maior que o divisor (essa subtração pode ser realizada pelo método de complemento), para se obter cada um dos algarismos do quociente.

Em face da complexidade dos processos e a pouca utilidade prática em computação, não serão apresentados mais detalhes nem exemplos de métodos para realização de operações de multiplicação e divisão com valores representados em decimal.

7.6 UM POUCO MAIS DE DETALHE

Neste item vamos apresentar alguns aspectos adicionais relativos à representação e à aritmética computacional em ponto flutuante e que poderá atender à expectativa de alguns leitores, bem como à necessidade e curiosidade de outros.

7.6.1 Sobre a Representação em Ponto Flutuante

Já foi mencionada anteriormente neste capítulo a diferença no formato de representação de números em ponto fixo e em ponto flutuante, bem como as características de cada formato e suas diferentes aplicações. Neste item, pretendemos apresentar outros aspectos relativos ao formato de ponto flutuante que tornam essa representação versátil e de amplo espectro de números, como é o caso de valores reais.

O nome "flutuante" para o ponto (ou vírgula) decimal é bastante sugestivo, pois nessa representação realmente podemos fazer o ponto separador das partes inteira e fracionária de um número variar rapidamente apenas variando o valor do expoente. Assim, o número decimal 1253 pode ser representado em notação científica como:

$N = 1253 = 0{,}1253 \times 10^{+04}$

No exemplo, o expoente é representado com apenas 2 algarismos.

Se, com os mesmos 2 algarismos do expoente, mudarmos seu valor de +04 para +14, o número muda bastante na sua grandeza, embora tenhamos mantido apenas os algarismos significativos 1253.

$N1 = 12530000000000 = 0{,}1253 \times 10^{+14}$

A faixa de valores que podem ser representados em ponto flutuante é determinada pela quantidade de algarismos escolhida para indicar o valor do expoente, conforme pudemos verificar no exemplo anterior.

Já a precisão do número é atribuição da quantidade de algarismos escolhida para indicar o valor da fração ou mantissa. Neste caso, alguns exemplos podem mostrar melhor essa afirmação:

3,14 3,1416 3,141592

Estes valores possuem diferentes precisões em face da quantidade de algarismos na parte fracionária. Quanto maior a quantidade de algarismos, maior sua precisão.

Como poucos algarismos na parte do expoente crescem consideravelmente a faixa de representação, usam-se muito mais algarismos para indicar o valor da fração do que para indicar o valor do expoente. Relações mais comuns entre esses valores são:

8 bits para o expoente e 23 bits para a fração ou

11 bits para o expoente e 52 bits para a fração.

Pode-se verificar, de modo simples, como se pode representar uma faixa muito maior de valores usando-se a forma de ponto flutuante em vez da forma de ponto fixo.

Consideremos um sistema de computação que empregue palavra de 16 bits cujos formatos e faixa de representação de números em ponto fixo (sinal e magnitude) e em ponto flutuante são:

Ponto Fixo (Sinal e Magnitude)

Sinal	Magnitude
1 bit	15 bits

Faixa de representação (para inteiros):

$-(2^{15} - 1)$ até $+(2^{15} - 1)$ ou, em decimal, -32767 até $+32767$

Ponto Flutuante — formato para $N = \pm \text{Fração (F)} \times B^{\pm E}$

Sinal	Expoente	Fração
1 bit	6 bits	9 bits

Faixa de representação:

Considerando que o expoente possui 6 bits, sendo 1 para seu sinal e 5 para seu valor, temos que ele pode variar (o expoente) de:

$-(2^5 - 1)$ até $+ (2^5 - 1)$ ou seja: de -31 até 0 e até $+31$

Os números podem, então, variar em uma enorme faixa, com valores menores que 1 (números fracionários), que não estão contemplados na representação de ponto fixo. Isto é possível devido ao expoente poder ser negativo, de -31 até -1.

Temos, também, toda a faixa de valores representada pelo expoente positivo, de 0 até $+31$.

E, mais ainda, todas essas faixas valem para números positivos e negativos.

Assim, com a mesma quantidade disponível de bits, 16 em nosso exemplo, podemos representar muito maior quantidade de números em ponto flutuante do que em ponto fixo, especialmente números fracionários.

As linguagens de programação mais populares utilizam os dois formatos (ponto fixo e ponto flutuante), geralmente sendo o ponto fixo para representação de inteiros e ponto flutuante para representação de fracionários.

Qualquer dos métodos utilizados para representar números em computador (sinal e magnitude, complemento a 1, complemento a 2, ponto flutuante) requer atenção do programdor pelo fato de que algumas operações realizadas podem redundar em erro no resultado. Isto é devido à quantidade finita e limitada de algarismos utilizada nas máquinas (no nosso cotidiano podemos operar com números de qualquer quantidade de algarismos desde que tenhamos um meio de tamanho infinito para escrevê-los).

Números inteiros são menos problemáticos porque o programador pode criar uma faixa menor e mais definida de valores em seu programa, porém, com números reais (fracionários), o problema se torna muito maior na medida em que podemos criar infinitos valores entre qualquer faixa de valores.

Dessa forma, há sempre um problema de precisão e arredondamento em operações matemáticas com esse tipo de valores. O problema fica ainda maior se cada fabricante adotar características diferentes para sua representação em ponto flutuante (algumas inserindo erros durante os cálculos matemáticos). Por essa razão, o IEEE decidiu desenvolver um padrão de formato para as representações em ponto flutuante.

7.6.2 O Padrão IEEE-754, 1985

O padrão de representação em ponto flutuante denominado IEEE-754 foi desenvolvido pelo IEEE — Institute of Electrical and Electronic Engineers com a finalidade óbvia que a própria nomenclatura indica: padronizar os diferentes formatos criados pelos fabricantes de processadores e reduzir a possibilidade de erros.

O referido padrão foi divulgado a partir de 1985 e atualmente é utilizado pela quase totalidade dos fabricantes, uma extraordinária vantagem para os usuários, sempre às voltas com a despadronização em várias áreas e aspectos da computação.

O padrão IEEE tem três partes básicas: o sinal do número (S_N), o expoente (E) e a mantissa ou fração (F). A base de representação e de exponenciação é implícita, usando-se a base 2 (binária), não necessitando de indicação no formato do número.

O primeiro campo a partir da esquerda representa o sinal do número e é o mais simples. Consta de 1 bit, cujo valor zero (0) indica que o número é positivo e valor 1 indica números negativos.

O campo seguinte representa o expoente. Ele compreende valores positivos e negativos de expoente e, embora sejam sempre valores inteiros, a forma adotada não é sinal e magnitude nem complemento.

O método adotado para representar os valores positivos e negativos do expoente é denominado excesso de N (em inglês chama-se também *bias*). Consiste em somar-se um valor, N ou bias, ao valor real do expoente e armazenar este resultado no campo E, expoente, da representação. O valor N é calculado de acordo com a quantidade de bits do campo E, sendo 127 no caso de precisão simples e 1023 no caso de precisão dupla. Na realidade, temos:

$N = (2^E/2) - 1$, sendo E = quantidade de bits do campo expoente.

Assim, se, por exemplo, um determinado expoente é igual ao valor 0, para precisão simples ele será armazenado como 127, pois:

127 (valor armazenado no campo E) = 0 (valor real do expoente) + 127 (N ou bias)

Se encontrarmos um valor igual a 175 no campo E, o valor real do expoente será igual a 48, pois:

$175 - 127 = 48$.

Adiante mostraremos que os valores extremos de N (bias) são usados para representar valores especiais, isto é, campo expoente igual a 0 (binário 00000000) ou igual a decimal 255 (11111111) para precisão simples e decimal 2047 (binário 11111111111) para precisão dupla.

O terceiro e último campo é o da fração ou mantissa, que no padrão IEEE-754 é denominado *significando* (ou *significand*). Como já mencionamos anteriormente, ele representa a precisão do número, a parte dos algarismos significativos do número.

O padrão 754 calcula o significando de forma a se obter um valor com um bit a mais, visto que:

a) usa a forma normalizada, já apresentada anteriormente, isto é, o primeiro bit após a vírgula é diferente de zero;

b) devido a este fato, a fração sempre inicia por 0,1xxxxxxx (onde x serão os demais algarismos) e, por isso, o padrão deixa de representá-los, o 0 e o primeiro 1. Ele assume que eles existem mas não os representa.

Há três formatos definidos pelo padrão 754: precisão simples (single), que emprega 32 bits, precisão dupla (double), que usa 64 bits, e precisão estendida (extended), que emprega 80 bits para a representação de cada número. A precisão estendida é usada pela Intel internamente em seus co-processadores matemáticos (80x87) e, portanto, deve ser evitada pelos programadores e não será aqui analisada.

A Tabela 7.6 mostra as características dos formatos básicos.

Tabela 7.6 Características dos Formatos Básicos de Números em Ponto Flutuante, Representados no Padrão IEEE-754

	Precisão simples	**Precisão dupla**
Total de bits do número	32 bits	64 bit
Quant. bits para sinal	1 bit	1 bit
Bits para expoente	8	11
Bits para significando	23	52
Cálculo do expoente	Excesso de 127	Excesso de 1023
Faixa de representação em decimal	Aprox. 10^{-38} até 10^{+38}	Aprox. 10^{-308} até 10^{+308}

Valores especiais:

Zero — sendo o valor do número zero, sua representação não se faz diretamente, devido ao fato de o significando omitir o primeiro 1. Deste modo, o valor 0 para um número é representado pelo campo do expoente igual a 0 (todos os bits sendo 0) e o significando também 0. O sinal poderá ser 1 ou 0, o que é irrelevante pois zero não tem sinal.

Infinito — os valores + infinito e − infinito são representados assim: bit de sinal podendo ser 0 (para + infinito) ou 1 (para − infinito), campo expoente igual a 255 ou 1023 (todos os bits iguais a 1), conforme se use precisão simples ou dupla. O campo da mantissa usa todos os bits iguais a 0.

Indeterminado — se um resultado é um valor indeterminado (por exemplo, o resultado de infinito − infinito ou 0 vezes infinito), então sua representação será: bit de sinal igual a 1; campo do expoente com todos os bits iguais a 1 e campo do significando com primeiro bit à esquerda igual a 1 e os demais iguais a 0.

Não é um número (not a number ou NaN) — caso ocorra um erro de algum modo, então sua representação será: bit de sinal igual a 0, todos os bits do expoente iguais a 1 e a mantissa deverá ser um valor qualquer diferente de zero.

Valor não-normalizado (denormalized) — se o campo expoente é constituído de 0s, mas o campo da mantissa não é igual a 0s, então o valor representado não está normalizado, o qual não tem, por isso, o bit 1 mais à esquerda assumido. Desta forma, a representação será de um valor compreendido, para precisão simples, na faixa de:

$(-1)^s \times 0 \cdot m \times 2^{-126}$ sendo: s = bit de sinal e m = a mantissa ou fração armazenada

E para precisão dupla na faixa de:

$(-1)^s \times 0 \cdot m \times 2^{-1022}$

O padrão IEEE-754 também estabelece de forma direta e bem definida o resultado de operações com números especiais, o que é mostrado na Tabela 7.7.

Tabela 7.7 Resultado de Operações com Números Especiais

Operação	Resultado
Número/±Infinito	0
±Infinito × ±Infinito	±Infinito
±Número/0	±Infinito
±Infinito + ±Infinito	Infinito
Infinito − Infinito	Indeterminado
±Infinito/±Infinito	Indeterminado
±Infinito × 0	Indeterminado

Todos os elementos mencionados sobre os casos especiais podem ser resumidos na Tabela 7.8, que indica os valores dos 3 campos — sinal, expoente e significando (fração ou mantissa) — para cada caso:

Tabela 7.8 Quadro-resumo da Representação de Casos Especiais no Padrão IEEE-754

Sinal	Expoente	Fração	Valor do número
0	000...000	000....000	+0
0	000...000	000...001 ..... 111...111	Número positivo não-normalizado $0 \cdot m \times 2^{(-b+1)}$

Tabela 7.8 Quadro-resumo da Representação de Casos Especiais no Padrão IEEE-754 (continuação)

Sinal	Expoente	Fração	Valor do número
0	000...001 ..... 111...110	XXX...XXX	Número positivo normalizado $1 \cdot m \times 2^{(e-b)}$
0	111...111	000....000	+Infinito
0	111...111	000...001 ..... 111...111	NaN (não é um número)
1	000...000	000....000	−0
1	000...000	000...001 ..... 111...111	Número negativo não-normalizado $-0 \cdot m \times 2^{(-b+1)}$
1	000...001 ..... 111...110	XXX...XXX	Número negativo-normalizado $-1 \cdot m \times 2^{(e-b)}$
1	111...111	000....000	−Infinito
1	111...111	000...001 ..... 011...111	NaN
1	111...111	100...000	Indeterminado
1	111...111	100...001 ..... 111.111	NaN

EXERCÍCIOS

1) Utilizando k dígitos binários, determine quantos números não-negativos podem ser representados em: sinal e magnitude; complemento a 1; complemento a 2.

2) O código de representação de caracteres ASCII é de 7 bits, enquanto o código EBCDIC é de 8 bits. Mostre uma possível razão para a escolha de um código de 7 bits, em vez de um de 8 bits, que permite maior quantidade de representação de símbolos.

3) Descreva os aspectos básicos do código de representação de caracteres denominado Unicode.

4) Considere os valores abaixo, representados em complemento a 2:

a) 11100000   b) 11001100   c) 11101111

d) 10001110   e) 10111011   f) 10000001

Considerando que a palavra do computador tenha 8 bits de tamanho, obtenha o resultado das operações a seguir, indicando se ocorrer *overflow*:

a − d   b − e   c − f

5) Converta os seguintes valores decimais para os formatos de representação de números indicados ao lado de cada um:

a) +119 para S/M, com palavra de 8 bits
b) −77 para S/M, com palavra de 16 bits
c) −135 para C1, com palavra de 8 bits
d) +217 para S/M, com palavra de 16 bits
e) −143 para C2, com palavra de 12 bits
f) −227 para C2, com palavra de 16 bits

6) Considerando um sistema de computação cuja palavra é de 16 bits, indique a faixa de representação de valores inteiros se o sistema opera com valores em:

a) sinal e magnitude
b) complemento a 1
c) complemento a 2

7) O complemento a 2 de um valor binário N pode ser definido como:

C2 de N = $2^N - N$

Mostre que o C2 do C2 de um número é o próprio número.

8) Por que o emprego da aritmética de complemento é mais vantajosa que a aritmética de sinal e magnitude? Por que a de complemento a 2 é ainda mais vantajosa que a de complemento a 1?

9) Considerando um sistema que utilize aritmética em ponto flutuante, mostre qual dos campos representativos do número é responsável pela precisão daquele número.

10) Indique a faixa limite de representação de números considerando computadores com o tamanho indicado de palavra:

a) complemento a 1 — palavra de 16 bits
b) complemento a 2 — palavra de 16 bits
c) sinal e magnitude — palavra de 12 bits
d) complemento a 2 — palavra de 12 bits

11) Converta cada um dos números decimais relacionados na representação de complemento a 2, em sistema com palavra de 16 bits:

a) +14 b) +6954 c) −1543 d) +28481
e) −328 f) −32768 g) −8739 h) −32767

12) Considere a seguinte representação de ponto flutuante:

S	E	M
1	5	10 bits

S — sinal do número
E — representação do expoente em sinal e magnitude
M — mantissa normalizada
Base de exponenciação: 2

Converta os valores decimais a seguir na representação de ponto flutuante indicada:

a) +0,00565 b) −674,25 c) +46,5 d) −0,0245 e) +1260,32

13) Considere a representação de números em ponto flutuante definida no exercício anterior. Converta os números abaixo, representados em ponto flutuante (são mostrados em hexadecimal para reduzir a quantidade de algarismos), para sua forma decimal. Podem ser indicados apenas em notação científica.

a) E745 b) 3FC6 c) F320

14) Considere a representação em ponto flutuante definida pelo padrão IEEE-754, 1985, com precisão simples. Converta os seguintes valores para aquela forma de representação:

a) +0,0012675 b) −12657 c) −0,03568

15) Os seguintes valores binários estão representados em ponto flutuante da seguinte forma: bit mais significativo (à esquerda) indica sinal do número, segue-se a representação do expoente, em excesso de 63 e uma mantissa com 24 bits. Efetue sua normalização:

a) 0 1000001 000101110000000100000000

b) 0 1111000 000000001101111100000000

c) 1 0001110 100000001000000000000000

16) Considere um sistema cuja aritmética de ponto fixo é realizada em complemento a 2 e que possua palavra de 7 bits. Efetue as operações indicadas (usando aritmética de C2), apresentando o resultado de cada uma em binário e decimal e explicitando quando ocorrer *overflow*:

A = 0111100 B = 1110110 C = 0001111 D = 0010100 E = 1111110

a) A − B b) D + C c) −A + D
d) E − A e) −A − C f) B − D

17) Considere a representação em ponto flutuante definida na Fig. 7.11. Qual é o maior valor positivo que pode ser representado? E qual é o menor valor positivo?

18) Considerando uma determinada representação de números em ponto flutuante, que campo deverá ser modificado se desejarmos aumentar a precisão dos números representados?

19) Qual é o inteiro mais negativo que pode ser representado em sinal e magnitude em um sistema com palavra de 16 bits?

20) Por que em complemento a 2 existe uma representação a mais de números negativos que de números positivos?

21) Considere os valores binários abaixo:

A = 110011 B = 011000 C = 010111 D = 000011 E = 111100

Obtenha o valor de X após a execução das seguintes equações:

a) X = NOT (B OR A) XOR (NOT B OR A) AND (C AND NOT A)

b) X = C XOR (A OR (B AND C) OR (A XOR B) AND NOT B)

c) $X = (\overline{A \oplus B}) + (D + \overline{A \cdot \overline{B}}) \cdot \overline{C} \oplus E$

d) $X = A \cdot \overline{B} \cdot C \cdot (E + \overline{D \oplus B})$

e) $X = A + (B \cdot D) \cdot (E + C)$

22) Quais são os operadores lógicos que sempre satisfazem as seguintes equações:

A op A = 0

A op 1 = A

23) Converta os valores a seguir (estão em decimal) em representação em complemento a 2 e realize as operações aritméticas indicadas, considerando sempre uma palavra de 16 bits:

A = −345 B = −563 C = +239 D = −893

a) A − C b) B + D c) B − A d) C + B

24) Converta os seguintes números decimais para complemento a 1 e para complemento a 2 (empregue palavra de 16 bits):

a) +219 b) −774 c) −225 d) +117

25) Realize as seguintes multiplicações e divisões usando o sistema binário:

a) 15 × 13 b) 19/6 c) 18 × 14 d) 44/11

26) Considere os seguintes números já representados em complemento a 2:

A = 11100011 B = 11001110 C = 11001100 D = 11101000

Efetue os cálculos a seguir, utilizando aritmética de complemento a 2:

a) A − B b) C + B c) D − C d) B + D

27) Considerando a representação de ponto flutuante indicada no Exercício 12, converta os seguintes valores decimais para aquela representação:

a) −173 b) −219 c) +237 d) −318

8

Representação de Instruções

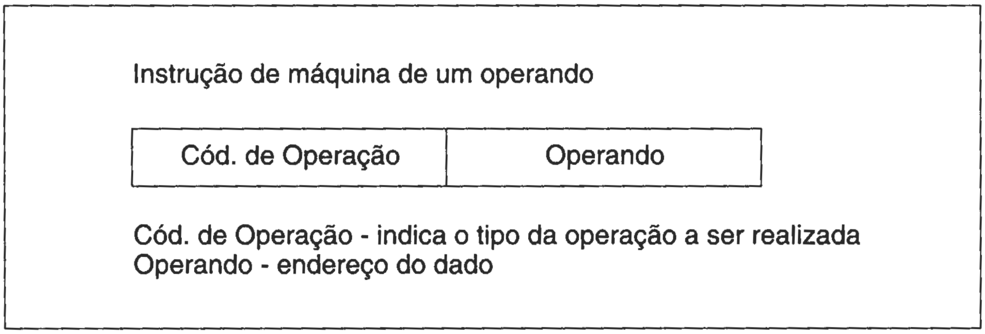
Conforme já mencionamos diversas vezes nos capítulos anteriores, o funcionamento básico de um computador está totalmente relacionado às operações primitivas que a UCP (o hardware) é capaz de realizar diretamente. Sabemos também que as referidas operações primitivas, básicas, têm sua execução efetivada através da realização, passo a passo, de uma seqüência de ações menores (denominadas microoperações). Esta seqüência constitui o algoritmo que conduz à execução da operação, o qual chamamos de instrução da máquina para realizar a dita operação.

As UCPs (os processadores) são fabricadas contendo internamente a programação para realização de uma grande quantidade de operações primitivas (ver Cap. 6), cada uma definida pela respectiva instrução de máquina, que é a formalização da operação em si.

Denomina-se *conjunto de instruções* (*instruction set*) de um processador esta quantidade de instruções que ele pode realizar diretamente.

Neste capítulo pretende-se apresentar um pouco mais de detalhes sobre as referidas instruções de máquina, ampliando assim as informações constantes dos capítulos anteriores. Entre esses detalhes destaca-se de modo importante o formato da instrução, isto é, o significado de cada um dos bits que constitui uma instrução de máquina.

Desde o Cap. 2, quando da apresentação dos principais componentes da Unidade Central de Processamento (UCP), têm sido mostradas, nos exemplos, instruções de máquina com um formato-padrão de um operando, conforme o diagrama da Fig. 8.1.



Cód. de Operação - indica o tipo da operação a ser realizada
Operando - endereço do dado

Figura 8.1 Exemplo de formato de instruções de um operando.

No entanto, o formato apresentado na figura não é o único utilizado nos sistemas de computação comerciais. Na realidade, há diversas formas de representar instruções, e vários modos da Unidade de Controle interpretar a busca do(s) dado(s).

Cada formato de instrução tem características próprias, com suas vantagens e desvantagens, podendo ser eficaz em certas aplicações e desaconselhável em outras.

Na prática, o conjunto de instruções definido para uma determinada UCP é sempre constituído de uma mistura de formatos diferentes, justamente para permitir a melhor aplicação em cada caso.

É interessante observar que há opiniões divergentes quanto à vantagem de se obter versatilidade com um numeroso conjunto de instruções de máquina.

O problema, neste caso, está na dificuldade dos compiladores em escolher a melhor opção de instrução para cada tipo de aplicação. A tendência é fazer com que os compiladores operem quase sempre com uma pequena quantidade de instruções; com isso, perderiam sentido tantas instruções pouco úteis (ou pouco utilizadas).

É conveniente repetir, neste ponto, alguns dos aspectos básicos sobre o funcionamento de um computador, especificamente referentes aos processadores (Unidade Central de Processamento), todos eles já apresentados no Cap. 6.

Os processadores são projetados com uma arquitetura definida com o único propósito de realizar operações básicas muito simples, tais como: somar dois valores, subtrair dois valores, mover um valor de um local para outro. A programação da seqüência de passos para realizar cada uma das mencionadas operações é inserida no processador durante o processo de sua fabricação, caracterizando a instrução de máquina.

Como a linguagem utilizada pelas máquinas para realizar suas tarefas é binária, também uma instrução de máquina deve ser representada nessa linguagem e, assim, ser formada por um conjunto de bits, que indica, conforme seu formato e programação, o que o processador deve realizar (qual a operação) e como realizar (a operação), além de ter que indicar com que dados a operação irá trabalhar (a localização do(s) dado(s)).

O projetista do processador escolhe, então, que operações aquele processador irá realizar e define, para cada uma delas, todos os detalhes de identificação e execução da operação, estabelecendo, assim, o formato de cada instrução de máquina.

Basicamente, uma instrução possui dois elementos:

— o que indica ao processador o que fazer e como fazer — denominado código de operação (C. Op.); e

— o que indica ao processador com que dado ou dados a operação irá se realizar — denominado operando (Op.).

Esses elementos são identificados para o processador por um grupo de bits específico, os quais, em conjunto, formam a instrução completa.

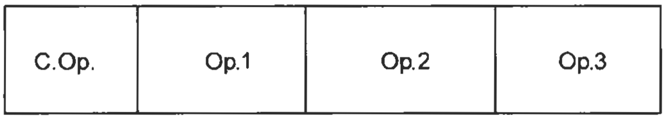
Toda instrução de máquina possui um código de operação específico e único para aquela tarefa. Este código, após ser decodificado durante o ciclo de execução da instrução, permitirá que a Unidade de Controle emita os sinais necessários, e previamente programados, para se efetivar a seqüência de passos de realização da operação indicada.

A quantidade de bits estabelecida para este campo da instrução (C. Op.) define o limite máximo de instruções que o processador poderá executar. Se, por exemplo, um determinado processador possui instruções de máquina, cujo C. Op. é um campo de 6 bits, então, este processador somente poderá realizar 64 instruções diferentes, dado que $2^6 = 64$.

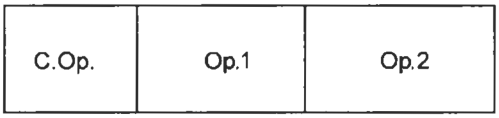
Além do código de operação, as instruções podem conter um ou mais campos denominados operando, cada um deles contendo informação sobre o dado a ser manipulado (o tipo da informação sobre o dado — seu valor ou o endereço de memória onde localizá-lo). A Fig. 8.2 mostra exemplos de alguns formatos típicos de instruções de máquina.

Para compreender, de modo ordenado, a representação de instruções em sistemas de computação, pode-se efetuar a análise do assunto segundo dois aspectos:

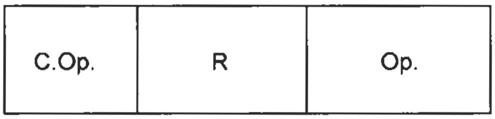
- quantidade de operandos;
- modo de interpretação do valor armazenado no campo operando (modo de endereçamento do dado).



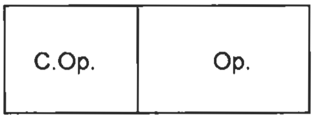
ADD Op.1, Op.2, Op.3
(a) Instrução de 3 operandos
(com memória)



MOVE Op.1, Op.2
(b) Instrução de 3 operandos
(com memória)



ADC R, Op.
(c) Instrução de 2 operandos, sendo um registrador



JCXZ Op.
(d) Instrução de 1 operando (desvio para memória)

Figura 8.2 Exemplos de formatos de instruções de máquina.

8.1 QUANTIDADE DE OPERANDOS

Desde os primeiros processadores concebidos até os atuais Pentium, Power PC, K7, Alpha e outros, os projetistas têm definido conjuntos de instruções dos mais variados tipos e formatos, de modo que, mesmo em um único e específico processador, as instruções tendem a ter formato diferente, isto é, tamanhos e campos diversos, conforme a operação que a instrução indica.

Assim, instruções que realizam operações aritméticas ou lógicas tendem a ter 2 operandos (embora uma instrução deste tipo devesse possuir 3 operandos para se tornar completa, como veremos mais adiante), algumas vezes 1 apenas (os outros ficam implícitos) e muito raramente 3 operandos. E assim por diante.

Um dos primeiros formatos de instrução idealizados foi incluído no sistema SEAC (Standard Eastern Automatic Computer), que ficou pronto em 1949 e possuía quatro operandos, conforme mostrado na Fig. 8.3.

Tal instrução (que não é mais utilizada — nem a máquina) era completa. Não só possuía indicação explícita da localização de todos os operandos (no caso, é claro, de se tratar de uma instrução que realiza uma operação aritmética), como também já trazia armazenado o endereço da próxima instrução.

No caso de um computador com memória de 2K células (endereços) e com instrução possuindo um código de operação de 6 bits (conjunto de 64 possíveis instruções), cada instrução teria um tamanho total de 50 bits.

C. Op.	Operando 1	Operando 2	Operando 3	End. próx. instrução

Operando n - endereço do dado ou o próprio valor do dado

Figura 8.3 Exemplo de formato de instrução de quatro operandos.

Se a memória tem 2K células, então cada endereço seria indicado por um número de 11 bits, pois:

$2^{11} = 2K$

Como cada campo operando contém o endereço de um dado e a instrução possui quatro operandos:

4 (operandos) × 11 (bits de endereços) + 6 (C. Op.) = 50 bits

Essa máquina poderia ter, por exemplo, uma instrução de soma do tipo (em Assembler):

ADD X,Y,Z,P, cuja descrição para execução seria: (Z) ← (X) + (Y), sendo P ← endereço da próxima instrução

Essa única instrução permitiria a execução da expressão:

C = A + B

podendo ser representada em linguagem Assembly como:

ADD A,B,C,P

Muitas considerações podem ser feitas a respeito das vantagens relativas à quantidade de operandos desse tipo de instrução, entre as quais podemos citar:

a) *completeza* — a instrução possui todos os operandos necessários à realização de uma operação aritmética, dispensando até instruções de desvio incondicional, pois o endereço de desvio consta no campo P;

b) *menor quantidade de instruções em um programa*, em comparação ao uso de instruções com menor quantidade de operandos, como veremos mais adiante.

Mas, apesar dessas vantagens, esse tipo de instrução tem uma grande desvantagem: a ocupação demasiada de espaço de memória, principalmente se atentarmos para o fato de que grande número de instruções de um programa não necessita de todos os três operandos (praticamente somente as instruções que tratam de operações matemáticas é que poderiam requerer 3 operandos).

Uma instrução de desvio incondicional, por exemplo, precisaria apenas do campo P (que conteria o endereço da próxima instrução, para onde se estaria querendo desviar), restando inúteis 33 bits da instrução (3 × 11 bits).

Outras instruções também deixam de usar todos os campos operandos. A instrução que transfere um valor da MP para a UCP (LOAD) necessita apenas de dois campos: um para o endereço do dado e outro para indicar o endereço da próxima instrução. Restariam inúteis dois campos ou 22 bits. Às vezes, este tipo de instrução poderia requerer outro operando, para indicar o destino do dado na UCP, se considerarmos que o local de destino (registrador) poderia ser um dentre vários. No exemplo, consideramos que o LOAD seria realizado armazenando o dado em um registrador especial e único, o acumulador, prescindindo, assim, de indicação explícita na instrução.

Um dos fatores mais importantes no projeto de uma UCP consiste na escolha do tamanho das instruções. Essa escolha depende de várias características da máquina, tais como:

- tamanho da memória;
- tamanho e organização das células da MP;
- velocidade de acesso;
- organização do barramento de dados.

O ponto crucial a ser analisado por ocasião do projeto reside na comparação entre dois fatores antagônicos: economia de espaço × conjunto completo e poderoso de instruções.

Um bom conjunto de instruções requer muitas instruções (para atender a diferentes tipos de aplicações), o que implica definir muitos códigos de operação (um para cada instrução, é claro) e, conseqüentemente, mais bits para o campo código de operação.

O aumento da quantidade de bits do campo C. Op. acarreta aumento do tamanho da instrução e, principalmente, aumento da tarefa do decodificador durante a execução do ciclo de instrução (lembre-se da explicação sobre o emprego do decodificador de códigos de operação, no Cap. 6).

Além disso, um grande conjunto de instruções pode indicar que elas sejam mais completas e, nesse caso, há necessidade de muitos bits na instrução, já que haverá diversos campos de operandos.

No entanto, quanto mais bits a instrução possui, mais memória se consome para armazená-la. Isso é contrário ao desejo de economia de espaço de armazenamento, visando à redução de custos (mesmo atualmente, com a redução dos preços dos dispositivos eletrônicos, memória é sempre um elemento caro).

O problema é de tal ordem que há uma corrente de pesquisadores e fabricantes que se utiliza de outra tecnologia para definir a arquitetura dos processadores, buscando economia e eficiência nessa área de especificação de conjunto de instruções. Esta arquitetura diferente, conhecida como RISC — Reduced Instruction Set Computer (Computador com Conjunto Reduzido de Instruções), será analisada em separado, no Cap. 11.

Continuando, pode-se obter a desejada economia, sem comprometer a flexibilidade das instruções e, como possível solução do problema, efetuar uma redução na quantidade de operandos nas instruções.

Se, por exemplo, na instrução mostrada na Fig. 8.3 fosse retirado um operando, então, os 44 bits seriam usados para três operandos, o que, mantido o mesmo tamanho da palavra, daria para se acessar endereços de 14 bits ($2^{14} = 16K$) em vez dos 2K anteriores. Além disso, os 2 bits restantes poderiam servir para aumentar o campo código de operação para 8 bits (aumentaria o conjunto de instruções para 256).

Outra possibilidade seria reduzir o tamanho total da instrução, melhorando o uso da memória e permitindo maiores programas.

Na prática, a busca de instruções menores redundou inicialmente na retirada do campo P. Isto foi possível através da concepção de uma técnica mais aperfeiçoada de obter, de forma automática, o endereço da próxima instrução. Esta técnica consistiu na criação de um registrador especial na UCP, cuja função indica o endereço da próxima instrução (sendo automaticamente incrementado, está sempre indicando novo endereço). Trata-se, como já vimos no Cap. 6, do CI — Contador de Instrução (ou em inglês, MAR — Memory Address Register).

8.1.1 Instruções com Três Operandos

Uma instrução que trata da execução de uma operação aritmética com dois valores requer, naturalmente, a indicação explícita da localização desses valores. Quando eles estão armazenados na MP, o campo operando deve conter, então, o endereço de cada um deles, o que indica a necessidade de 2 campos operandos. Além disso, se se trata de uma soma de valores é natural imaginar que o sistema deve ser orientado para armazenar o resultado em algum local e, assim, deve haver um terceiro campo operando, para indicar o endereço da MP onde será armazenado o resultado.

A Fig. 8.4 apresenta o formato básico de uma instrução de três operandos. Pode-se estabelecer, por exemplo, que os campos Operando 1 e Operando 2 representem o endereço de cada dado utilizado como operando em uma operação aritmética ou lógica e que o campo Operando 3 contenha o endereço para armazenamento do resultado dessas operações.

As instruções de três operandos, empregadas em operações aritméticas, podem ser do tipo:

ADD A,B,X	$(X) \leftarrow (A) + (B)$
SUB A,B,X	$(X) \leftarrow (A) - (B)$
MPY A,B,X	$(X) \leftarrow (A)\ (B)$
DIV A,B,X	$(X) \leftarrow (A)\ (B)$

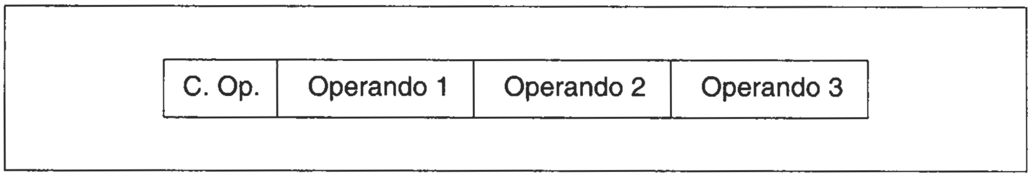


Figura 8.4 Exemplo de formato de instrução de três operandos.

Para exemplificar sua utilização, consideremos que um programa escrito em linguagem de alto nível contenha o comando mostrado a seguir, o qual calcula o valor de uma expressão algébrica:

$$X = A * (B + C * D - E/F) \tag{8.1}$$

Como resultado do processo de compilação (ver Cap. 9), o referido comando será convertido em instruções de máquina que, em conjunto, representam um programa com resultado idêntico ao do comando já apresentado.

Para simplificar e melhorar nosso entendimento, vamos utilizar as instruções da linguagem Assembly para montar o programa equivalente, em vez de criarmos instruções em linguagem binária:

A seqüência do algoritmo para resolver a equação é a seguinte, considerando as regras matemáticas usuais:

1) Inicialmente, resolver as operações internas aos parênteses.
2) Dentre as operações existentes, a primeira a ser realizada é a multiplicação de C por D (o resultado é armazenado em uma variável temporária, T1) e, em seguida, a divisão de E por F (resultado em uma variável temporária, T2) — prioridade dessas operações sobre soma e subtração.
3) Posteriormente, efetua-se a soma de B com T1.
4) Subtrai-se T2 do resultado dessa soma.
5) Finalmente, multiplica-se A por esse último resultado e armazena-se em X.

Considerando as instruções Assembly de 3 operandos, anteriormente definidas, podemos construir o seguinte programa equivalente ao comando exemplificado:

```
MPY   C,D,TI   ; multiplicação de C e D, resultado em T1 (item 2)
DIV   E,F,T2   ; divisão de E por F, resultado em T2 (item 2)
ADD   B,TI,X   ; soma de B com T1; resultado em X (item 3)
SUB   X,T2,X   ; subtração entre X e T2, resultado em X (item 4)
MPY   A,X,X    ; multiplicação de A por X, resultado em X (item 5)
```

Utilizaram-se duas variáveis temporárias, T1 e T2. Todas as letras usadas (A, B, C, D, E, F, T1, T2) representam endereços simbólicos de memória. A ordem de execução foi a normal: da esquerda para a direita, de acordo com a prioridade matemática.

Observemos que há operandos com endereços iguais, o que é um desperdício de espaço de memória. Também deve ser observado que o número de instruções é igual ao de operações; isso sempre acontecerá com instruções de 3 operandos, pois cada uma delas resolve por completo uma operação.

Ainda que se tenha reduzido a quantidade de operandos (de quatro para três), continuamos a consumir demasiado espaço de memória para a efetiva utilização dos operandos (continua a haver muitas instruções que não requerem todos os campos de operandos).

Instruções de máquina de 3 operandos são raramente encontradas em conjuntos de instruções dos atuais processadores existentes no mercado, devido principalmente ao seu grande tamanho.

8.1.2 Instruções com Dois Operandos

No exemplo anterior, pudemos observar que a maioria das instruções exige apenas dois endereços (o outro é repetido). Considerando a importância do problema de economia de espaço de armazenamento, foram criadas instruções com dois campos de operandos, como:

```
ADD    A,B    (A) ← (A) + (B)
```

As demais operações aritméticas seriam realizadas com instruções de formato igual. Na realidade, o conjunto de instruções aritméticas de 2 operandos poderia ser do tipo a seguir indicado:

```
ADD    Op.1,Op.2    (Op.1) ← (Op.1) + (Op.2)
SUB    Op.1,Op.2    (Op.1) ← (Op.1) - (Op.2)
MPY    Op.1,Op.2    (Op.1) ← (Op.1) * (Op.2)
DIV    Op.1,Op.2    (Op.1) ← (Op.1) /(Op.2)
```

Nesse caso, o conteúdo da posição de memória, cujo endereço está indicado em Op.1 (valor do primeiro operando) será destruído com o armazenamento, naquele endereço, do resultado da operação. Pode-se evitar, quando necessário, essa destruição, "salvando-se" o valor da variável, antes da execução da instrução.

Esse "salvamento" de variável pode ser realizado por uma nova instrução:

```
MOVE    A,B    (A) ← (B)
```

Com essas instruções de dois operandos, o comando correspondente à Eq. (8.1) poderia ser convertido, para execução, no programa Assembler apresentado a seguir, ainda de acordo com a seqüência apresentada no item anterior:

```
MPY     C,D     ; multiplicação de C por D, resultado em C (item 2)
DIV     E,F     ; divisão de E por F, resultado em E (item 2)
ADD     B,C     ; soma de B com C, resultado em B (item 3)
SUB     B,E     ; subtração entre B e E, resultado em B (item 4)
MPY     A,B     ; multiplicação de A por B, resultado em A (item 5)
MOVE    X,A     ; armazenamento do resultado final, A, em X
```

Note, agora, que a seqüência contém uma instrução a mais que o número de operações da expressão; e, também, podemos observar que foram destruídos os valores armazenados nos endereços correspondentes às variáveis A, B, C e E.

É sempre provável que se empregue, em um programa, uma variável mais de uma vez. Para evitar que uma determinada variável tenha seu valor destruído devido ao armazenamento de um resultado parcial no endereço correspondente, pode-se usar algumas variáveis temporárias e instruções MOVE, com o propósito de preservar todos os valores de variáveis.

Assim, a execução da instrução

```
MPY     C,D
```

acarretaria a destruição do valor da variável C, já que a descrição da instrução orienta a soma do valor armazenado no endereço indicado no campo (Op.1) que, neste caso, é correspondente à variável C, com o valor armazenado no endereço indicado no campo (Op.2), que, no caso, é correspondente à variável D, e que o resultado obtido (valor C + D) seja armazenado na MP no endereço indicado no mesmo campo (Op.1), que era o de C.

Para evitar essa destruição, o programa anterior poderia ser alterado para o seguinte:

```
MOVE    X,C     ; mover cópia de C para o endereço X
MPY     X,D     ; multiplicar X (cópia de C) por D. O resultado será armazenado em X e não mais
                  em C e, assim, o valor da variável não é destruído (item 2).
MOVE    T1,E    ; mover cópia de E para o endereço T1
DIV     T1,F    ; dividir T1 (cópia de E) por E. O resultado será armazenado em T1 e não mais
                  em E (item 2).
```

```
ADD     X,B     ; somar X por B, resultado em X (item 3)
SUB     X,T1    ; subtrair T1 de X, resultado em X (item 4)
MPY     X,A     ; multiplicar A por X, resultado final em X (item 5)
```

8.1.3 Instruções com Um Operando

Considerando as vantagens obtidas com a redução da quantidade de operandos (instruções menores), foram também criadas instruções de apenas um operando (instruções do tipo usado no Cap. 5).

Com esse tipo, o acumulador (ACC) é empregado como operando implícito (não é necessário especificar seu endereço na instrução, pois só há um ACC), guardando o valor de um dos dados e, posteriormente, o valor do resultado da operação.

```
ADD Op.   ACC ← ACC + (Op.)
SUB Op.   ACC ← ACC − (Op.)
MPY Op.   ACC ← ACC (Op.)
DIV Op.   ACC ← ACC (Op.)
```

Com o propósito de permitir a transferência de dados entre o ACC e a MP, foram criadas duas novas instruções:

```
LDA Op.   que significa   ACC ← (Op.)
STA Op.                   (Op.) ← ACC
```

Ainda o mesmo comando mostrado na Eq. (8.1) poderia ser convertido no programa Assembler, mostrado a seguir, constituído de instruções de um operando (sem destruirmos valor algum das variáveis).

```
LDA C
MPY D
STA X
LDA E
DIV F
STA T1
LDA B
ADD X
SUB T1
MPY A
STA X
```

A comparação entre os diferentes programas (para instruções com 3 operandos, com 2 e com 1 operando) é mostrada na Tabela 8.2, a qual inclui, também, a quantidade de memória despendida com cada programa, bem como a quantidade de acessos à memória (para ciclos de leitura ou de escrita) em cada caso.

Para calcular a quantidade de bits gastos em cada programa foram considerados um tamanho de código de operação de 8 bits (para todos os 3 tipos de instrução) e uma MP com capacidade de armazenamento de 1M células, sendo, portanto, cada campo operando de 20 bits (endereço do dado), visto que $2^{20} = 1M$.

A Tabela 8.1 apresenta o tamanho em bits de cada tipo de instrução, bem como a quantidade de ciclos de memória (de acessos) que são consumidos em cada uma delas.

Os programas mostrados na Tabela 8.2 indicam que, em termos de gasto de memória, instruções de 3 operandos não servem, razão por que foram abandonados pelos fabricantes há muito tempo. Usualmente, as instruções de dois operandos são mais bem empregadas em operações matemáticas (aritméticas e lógicas), utilizando-se também instruções de 1 operando em outros casos, como instruções de desvio.

Tabela 8.1 Tamanho e Consumo de Tempo de Execução de Instruções de 3, de 2 e de 1 Operando

Instrução de 3 operandos —	C.Op. = 8 bits + 3 operandos de 20 bits cada um = 68 bits Ciclos de memória = 4 (um para buscar a instrução e 3 para cada operando)
Instrução de 2 operandos —	C.Op. = 8 bits + 2 operandos de 20 bits cada um = 48 bits Ciclos de memória = 4 (um para buscar a instrução e 3 para cada operando)
Instrução de 1 operando —	C.Op. = 8 bits + 1 operando de 20 bits = 28 bits Ciclos de memória = 2 (um para buscar a instrução e 1 para o operando)

Tabela 8.2 Programas Assembly para Solucionar a Equação: X = A * (B + C * D − E/F)

Com instruções de 3 operandos	Com instruções de 2 operandos (sem salvamento)	Com instruções de 2 operandos (com salvamento)	Com instruções de 1 operando

MPY C, D, T1
DIV E, F, T2
ADD B, T1, X
SUB X, T2, X
MPY A, X, X
	MPY C, D
	DIV E, F
	ADD B, C
	SUB B, E
	MPY A, B
	MOVE X, A
		MOVE X, C
		MPY X, D
		MOVE T1, E
		DIV T1, F
		ADD X, B
		SUB X, T1
		MPY X, A
			LDA C
			MPY D
			STA X
			LDA E
			DIV F
			STA T1
			LDA B
			ADD X
			SUB T1
			MPY A
			STA X

Espaço: 340 bits Tempo: 20 acessos	Espaço: 288 bits Tempo: 24 acessos	Espaço: 336 bits Tempo: 28 acessos	Espaço: 308 bits Tempo: 22 acessos

Na verdade, há ainda muita controvérsia em relação ao projeto e implementação do conjunto de instruções de um processador, não existindo, de modo nenhum, unanimidade para os diversos tópicos, tais como tamanho da instrução, formato e significado do campo operando.

Instruções de poucos operandos ocupam menos espaço de memória e tornam o projeto do processador mais simples em virtude das poucas ações que elas induzem a realizar. No entanto, o programa gerado em binário é algumas vezes maior devido ao aumento da quantidade de instruções (não é o caso do exemplo apresentado na Tabela 8.2).

Instruções de 1 operando são simples e baratas de implementar, porém somente empregam um único registrador (o ACC) e, com isso, reduzem a flexibilidade e velocidade de processamento (o emprego de mais de um registrador acelera o processamento devido à velocidade de transferência desses dispositivos).

Instruções com mais de 1 operando podem usar tanto endereços de memória como registradores em seu formato, outro item de discussão.

8.2 MODOS DE ENDEREÇAMENTO

No Cap. 6, descrevemos o formato básico de instruções de máquina e o ciclo de execução de cada instrução, concluindo que:

a) O endereçamento de uma instrução é sempre realizado através do valor armazenado no contador de instrução (CI). Todo ciclo de instrução é iniciado pela transferência da instrução para o RI (usando-se o endereço contido no CI).

b) Toda instrução consiste em uma ordem codificada (código de operação) para a UCP executar uma operação qualquer sobre dados. No contexto da interpretação de uma instrução, o dado pode ser um valor numérico, um caractere alfabético, um endereço (instrução de desvio).

c) A localização do(s) dado(s) pode estar explicitamente indicada na própria instrução, por um ou mais conjuntos de bits, denominados *campo do operando*, ou implicitamente (quando o dado está armazenado no acumulador, que não precisa ser endereçado por ser único na UCP).

Todos os exemplos apresentados até esse ponto definiram o campo operando como contendo o endereço da MP onde está localizado o dado referido na instrução. No entanto, essa não é a única maneira de indicar a localização de um dado, havendo outros *modos de endereçamento*.

A existência de vários métodos para localizar um dado que está sendo referenciado em uma instrução se prende à necessidade de dotar os sistemas de computação da necessária flexibilidade no modo de atender aos diferentes requisitos dos programas.

Conforme será demonstrado a seguir, há instruções em que é ineficiente usar o dado armazenado na MP, como, por exemplo, no caso de um contador, o qual tem um valor fixo inicial e, durante a execução do programa, é sistematicamente atualizado. Nesse caso, melhor seria se o referido contador (dado) fosse inicialmente transferido para um registrador disponível na UCP e lá permanecesse (sendo diretamente atualizado na UCP) até o final da execução do programa, em vez de ir da MP para a UCP e vice-versa (para atualização de seu valor), o que acarreta um considerável gasto de tempo para os repetidos ciclos de leitura e gravação.

Por outro lado, a manipulação de vetores acarreta a necessidade de se estabelecer um método eficaz de endereçamento para variáveis que ocupam posições contíguas de memória. E assim por diante.

Dentre os diversos modos de endereçamento atualmente empregados, os principais são:

- imediato;
- direto;
- indireto;
- por registrador;
- indexado;
- base mais deslocamento.

8.2.1 Modo Imediato

O método mais simples e rápido de obter um dado é indicar seu próprio valor no campo operando da instrução, em vez de buscá-lo na memória. A vantagem desse método reside no curto tempo de execução da instrução, pois não gasta ciclo de memória para sua execução, exceto o único requerido para a busca da instrução.

Assim, o dado é transferido da memória juntamente com a instrução (para o RI), visto estar contido no campo operando da instrução.

Esse modo, denominado imediato, é útil para inicialização de contadores (um valor sempre fixo em toda execução do mesmo programa); na operação com constantes matemáticas; para armazenamento de ponteiros em registradores da UCP; ou para indicação da quantidade de posições em que um determinado número será deslocado para a direita ou para a esquerda (em operações de multiplicação e divisão).

Uma de suas desvantagens consiste na limitação do tamanho do campo operando das instruções, o que reduz o valor máximo do dado a ser manipulado.

Outra desvantagem é o fato de que, em programas repetidamente executados, com valores de variáveis diferentes a cada execução, esse método acarretaria o trabalho de alteração do valor do campo operando a cada execução (dado de valor diferente).

Praticamente, todo computador possui uma ou mais instruções que empregam o modo de endereçamento imediato: para instruções de desvio; de movimentação de um dado; para operações aritméticas com uma constante, etc.

Por exemplo, os processadores da família x86 possuem, entre outras, a instrução:

MOV R, Op., que pode ser assim usada: MOV AL, 22H (copiar o valor hexadecimal 22 para o registrador AL, sendo o valor de tamanho igual a 1 byte).

Com esta mesma instrução é possível copiar um valor de 32 bits. Por exemplo: MOV EBX, 33445566H (copiar o valor hexadecimal 33445566 para o registrador de 32 bits EBX).

Exemplo 8.1

C. Op.	Operando
4 bits	8 bits

JMP Op. CI ← Op.

C. Op. = 1010 = hexadecimal A

Instrução: 101000110101 ou A35 (C. Op. = A e Operando = 35) ———— Armazenar o valor 35 no CI.

Exemplo 8.2

C. Op.	R	Operando	MOV R, Op. R ← Op. C. Op. = 0101 = hexadecimal 5
4 bits	4 bits	8 bits	

Instrução: 0101001100000111 ou 5307 (C. Op. = 5, R = 3 e Operando = 07) ———— Armazenar o valor 07 no registrador de endereço 3 (R3).

8.2.2 Modo Direto

Nesse método, o valor binário contido no campo operando da instrução indica o endereço de memória onde se localiza o dado. Tem sido o modo empregado em nossos exemplos anteriores.

O endereço pode ser o de uma célula onde o dado está inteiramente contido ou pode indicar o endereço da célula inicial, quando o dado está armazenado em múltiplas células.

É também um modo simples de acesso, pois requer apenas uma referência à MP para buscar o dado, sendo, porém, mais lento que o modo imediato, devido naturalmente à referência à memória.

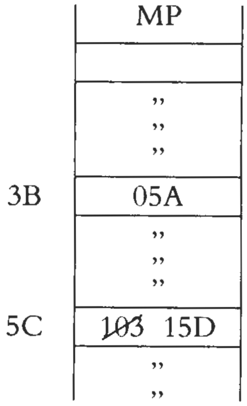
Quando um dado varia de valor a cada execução do programa, a melhor maneira de utilizá-lo é, inicialmente, armazená-lo na MP (do dispositivo de entrada para a memória). O programa, então, usa o dado pelo modo direto, em que a instrução indica apenas o endereço onde ele se localiza.

Uma possível desvantagem desse processo está na limitação de memória a ser usada, conforme o tamanho do campo operando. Isto é, se o campo tiver um tamanho, por exemplo, de 12 bits, com o emprego do modo direto, somente se pode acessar as células de endereço na faixa de 0 a 4095 (decimal), correspondentes aos valores binários 000000000000 a 111111111111.

Atualmente, como o espaço de endereçamento de MP vem crescendo bastante (usa-se MP com espaço de endereçamento da ordem de 32M células, 64MB e até 256MB), não é desejável criar instruções com campo operando de tantos bits — para endereçar 64M células seriam necessários 26 bits para endereço direto.

Os processadores da família Intel x86 possuem instruções no modo direto, uma das quais, do tipo MOV R, Op., que pode ser implementada copiando até 4 células contíguas de memória, 32 bits, para um registrador.

Exemplo 8.3



a)

C. Op.	Operando
4	8 bits

C. Op. = 7 LDA Op. ACC ← (Op.)

Após a execução da instrução, o ACC conterá o valor 05A.

b)

C. Op.	Op. 1	Op. 2
4	8	8 bits

C. Op. = B ADD Op.1, Op.2 (Op.1) ← (Op.1) + (Op.2)

Instrução: B55C3B

Somar o dado de valor binário 000100000011 ou hexadecimal 103 armazenado na célula de endereço hexadecimal 5C (Op. 1) com o dado de valor binário 000001011010 ou hexadecimal 05A armazenado na célula de endereço 3B (Op. 2) e armazenar o resultado (15D) na célula de endereço hexadecimal 5C (Op. 1).

8.2.3 Modo Indireto

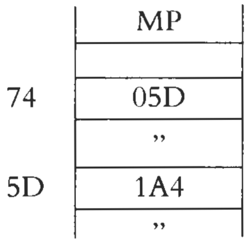
Nesse método, o valor binário do campo operando representa o endereço de uma célula; mas o conteúdo da referida célula não é o valor de um dado (como no modo direto), é um outro endereço de memória, cujo conteúdo é o valor do dado.

Assim, há um duplo endereçamento para o acesso a um dado e, conseqüentemente, mais ciclos de memória para buscar o dado, comparativamente com os métodos já apresentados.

O endereço intermediário (conteúdo da célula endereçada pelo valor do campo operando) é conhecido como *ponteiro*, pois indica a localização do dado ("aponta" para o dado). O conceito de ponteiro de dado é largamente empregado em programação.

Com esse processo, elimina-se o problema do modo direto, de limitação do valor do endereço do dado, pois estando o endereço armazenado na memória (pode ocupar uma ou mais células), ele se estenderá ao tamanho necessário à representação do maior endereço da MP do sistema de computação em uso.

Exemplo 8.4



C.Op.	Operando
4	8 bits

C.Op. = 4 LDA Op. ACC ← ((Op.))
Instrução: 474

74 é o endereço da célula cujo conteúdo (5D) é o endereço do dado (1A4). Após a execução, o valor 1A4 estará armazenado no ACC.

Há uma variação pouco empregada desse método, em que, em vez de um nível de indireção, são usados múltiplos níveis. Em lugar do valor do dado (caso de apenas um nível de indireção), armazena-se novo endereço, que acessa uma célula, que pode conter outro endereço, e assim sucessivamente.

Uma das implementações dessa variação do modo indireto prevê a inclusão de um bit especial na instrução, o qual representa a continuação ou não do endereçamento. Se o valor desse bit for igual a 0, o endereçamento prossegue com novo acesso, enquanto o valor 1 significa o encerramento do processo, isto é, o endereço final do dado.

Este método é atualmente pouco empregado devido à quantidade de acessos à memória e sua complexidade para programadores em linguagem Assembly.

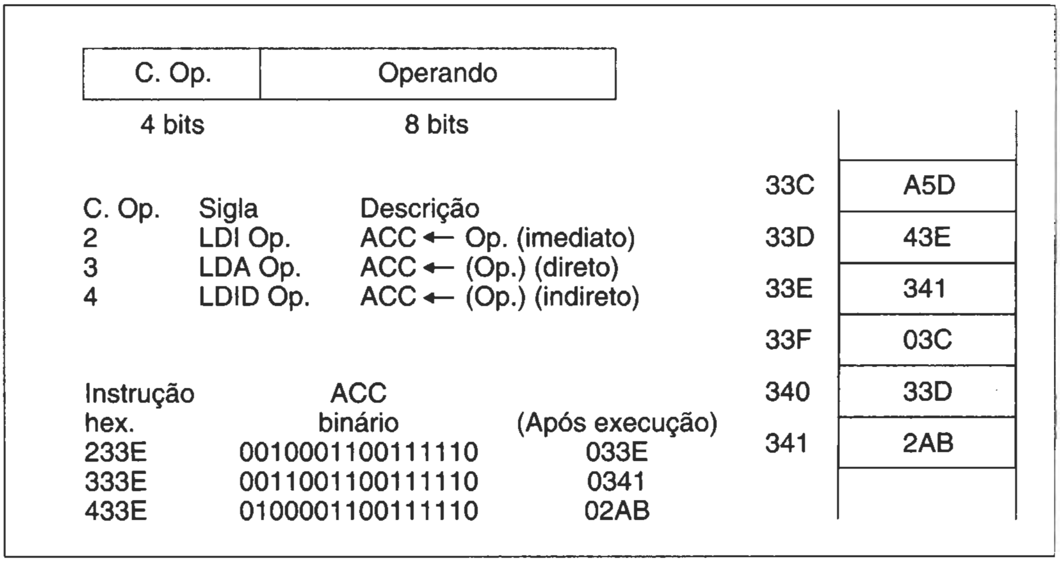
Um dos possíveis usos (no passado) para o modo indireto é na manutenção de ponteiros de dados. Se tivermos uma relação de dados a serem movimentados para novas posições de memória (caso, por exemplo, de elementos de vetores) usando o modo indireto, basta apenas modificar o valor da célula endereçada no primeiro nível (campo do operando), isto é, modificar o endereço de acesso ao dado, sem alterar o valor do campo operando.

Como já mencionado anteriormente, a grande desvantagem desse método é, obviamente, a maior quantidade de ciclos de memória requerida para completar o ciclo da instrução, pois, para se acessar um dado no modo indireto, é necessário efetuar dois acessos à memória (um para buscar o endereço do dado e outro para efetivamente buscar o dado).

Ainda sobre o processo de acesso do modo indireto, deve-se ter atenção para o fato de que, embora seja possível localizar qualquer endereço de MP (porque o endereço de efetivo acesso ao dado não está mais contido no campo operando da instrução, como no modo direto), em um dado instante somente se pode acessar endereços até o limite do campo operando, pois esse, contendo o endereço de acesso ao ponteiro do dado (endereço do dado), limita o tamanho do maior valor de endereço.

Observações

1) Há dois métodos de indicação do modo de endereçamento de instruções:
    a) Cada código de operação estabelece não só o tipo da instrução como também o modo de endereçamento. A Fig. 8.5 apresenta um exemplo desse método.
    b) A instrução possui um campo específico para indicar o modo de endereçamento; nesse campo, consta um código binário correspondente ao modo desejado. A Fig. 8.6 apresenta um exemplo desse método.
2) Comparando-se as características dos três modos de endereçamento apresentados, pode-se observar que:
    a) O modo *imediato* não requer acesso à MP para buscar o dado (exceto, é claro, para a busca da instrução); o modo *direto* requer um acesso e o modo *indireto*, pelo menos dois acessos para busca do dado na MP.



C. Op.	Sigla	Descrição
2	LDI Op.	ACC ← Op. (imediato)
3	LDA Op.	ACC ← (Op.) (direto)
4	LDID Op.	ACC ← (Op.) (indireto)

Instrução hex.	binário	ACC (Após execução)
233E	0010001100111110	033E
333E	0011001100111110	0341
433E	0100001100111110	02AB

33C	A5D
33D	43E
33E	341
33F	03C
340	33D
341	2AB

Figura 8.5 Modo de endereçamento implícito no código de operação.

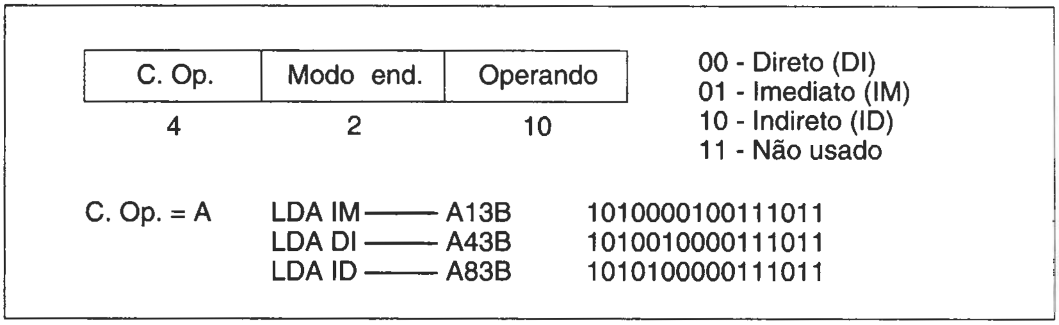


Figura 8.6 Exemplo de instrução com indicação explícita do modo de endereçamento.

b) Quanto ao tempo de execução das instruções, as que usam o modo *imediato* são mais rápidas, seguidas das que usam o modo *direto* e, finalmente, as que usam o modo *indireto* são executadas mais lentamente. A velocidade de execução é diretamente proporcional à quantidade de acessos despendida em cada ciclo da instrução.

A Fig. 8.7 apresenta um resumo comparativo da definição, vantagens e desvantagens de cada modo de endereçamento.

8.2.4 Endereçamento por Registrador

Esse método tem característica semelhante aos modos direto e indireto, exceto que a célula (ou palavra) de memória referenciada na instrução é substituída por um dos registradores da UCP. Com isso, o endereço mencionado na instrução passa a ser o de um dos registradores, e não mais de uma célula da MP.

A primeira vantagem, logo observada, consiste no menor número de bits necessários para endereçar os registradores, visto que esses existem em muito menor quantidade que as células de memória. Isto reduz o tamanho geral da instrução.

Um computador que tenha uma UCP projetada com 16 registradores requer apenas 4 bits para endereçá-los (cada um dos 16 registradores tem um endereço de 4 bits, de 0 a F_{16}, por exemplo); no caso de endere-

Modo de endereçamento	Definição	Vantagens	Desvantagens
Imediato	O campo operando contém o dado.	Rapidez na execução da instrução.	Limitação do tamanho do dado. Inadequado para o uso com dados de valor variável.
Direto	O campo operando contém o endereço do dado.	Flexibilidade no acesso a variáveis de valor diferente em cada execução do programa.	Perda de tempo, se o dado é uma constante.
Indireto	O campo operando contém o endereço do dado.	Manuseio de vetores (quando o modo indexado não está disponível). Uso como "ponteiro".	Muitos acessos à MP para execução.

Figura 8.7 Quadro demonstrativo das características dos modos de endereçamento.

çamento de células de MP, há necessidade de 20 ou mais bits para indicar o endereço de cada uma das células.

Outra vantagem está no próprio emprego do dado, que passa a ser armazenado em um meio (registrador) cujo acesso é muito mais rápido que o acesso à memória.

Para mostrar, de modo mais objetivo, a utilidade e as vantagens do uso de registradores no endereçamento de instruções, vamos considerar a execução do conjunto de instruções mostrado a seguir (pode ser parte de um programa), através de dois modos diferentes: *com emprego do modo de endereçamento por registrador* e *sem emprego desse modo de endereçamento.*

```
DO I = 1 TO 100
READ A, B
X = A + B
END
```

O trecho de programa descrito executa 100 vezes o mesmo tipo de ação: ler dois valores e somá-los. Para implementar sua execução direta é necessário definir uma variável inteira (chamamos de *contador*); após a execução de cada conjunto de instruções de leitura dos dados (Get) e de soma, o contador é incrementado de 1, até atingir o valor 100, quando a execução do trecho de programa se completa.

Em linguagem Assembly teríamos:

```
        GET L     ; ler valor do "loop" (no exemplo o valor é igual a 100)
        LDI 0     ; ACC ← 0
        SUBM L    ; ACC ← ACC − (L), no exemplo o valor é 100
In      STA I     ; (I) ← ACC (inicialmente I = 100)
        JZ Fim    ; se ACC = 0 vá para Fim
        GET A     ; ler valor do dado para o endereço A
        GET B     ; ler valor do dado para o endereço B
        LDA A     ; ACC ← (A)
        ADD B     ; ACC ← ACC + (B)
        STR X     ; (X) ← ACC
        LDA I     ;  ACC ← (I)
        INC       ; ACC ← ACC + 1 (no exemplo, estamos fazendo I + 1)
        JMP In    ; vá para In
Fim     HLT       ; parar
```

Como podemos observar, as instruções LDA I e STA I manipulam com a leitura e gravação de cada um dos 100 valores assumidos por I durante a execução do programa; gastam-se, com isso, 200 ciclos de memória (100 de leitura de I − LDA Op. e 100 de gravação do novo valor de I − STA Op.).

Vamos agora executar o mesmo trecho de programa de outra forma. Desta vez vamos utilizar o modo de endereçamento por registrador.

O objetivo é armazenar na UCP (em um dos registradores disponíveis) o valor inicial de 1 e efetuar a soma I = I + 1, diretamente na UCP (sem necessidade de acesso à MP, visto que o valor de I permanece armazenado em um dos registradores da UCP). Esse método irá economizar os 200 acessos gastos pelo processo anterior.

O novo trecho de programa em linguagem Assembly ficaria assim:

```
        LDI R1, 1        ; R1 ← 1 (o registrador recebe o valor inicial do contador, que é 1)
        LDI R2, 100      ; R2 ← 100 (o registrador recebe o valor final do contador, que é 100)
```

```
In    SUBR R1, R2    ; (R1) ← (R1) − (R2)
      JZ R, Fim      ; se R1 = 0 vá para Fim
      GET A          ; ler valor do dado para o endereço A
      GET B          ; ler valor do dado para o endereço B
      LDA A          ; ACC ← (A)
      ADD B          ; ACC ← ACC + (B)
      STR X          ; (X) ← ACC
      INC R1         ; (R1) ← ( R1) + 1 (no exemplo, estamos fazendo I + 1)
      JMP In         ; vá para In
Fim   HLT            ; parar
```

As instruções LDI R1, 1 e LDI R2, 100 armazenam em registradores, respectivamente, o valor inicial (1) e o valor final (100) do contador. No exemplo foram indicados os registradores R1 e R2, mas podem ser usados quaisquer registradores disponíveis.

As instruções SUBR R1, R2 e INC R1 serão executadas manipulando-se os valores armazenados dentro da UCP, nos registradores escolhidos, R1 e R2, cujos endereços constam da instrução (campo R); não há, portanto, acesso à MP.

Outra vantagem do emprego do modo de endereçamento por registrador — economia de bits nas instruções — pode ser verificada pela seguinte comparação:

Instrução com uso de registradores

← 16 bits →

C. Op.	R1	R2
8 bits	4 bits	4 bits

Instrução com acesso a 64K células de MP

← 40 bits →

C. Op.	Op. 1	Op. 2
8 bits	16 bits	16 bits

Há duas maneiras de empregar o modo de endereçamento por registrador:

- modo por registrador direto;
- modo por registrador indireto.

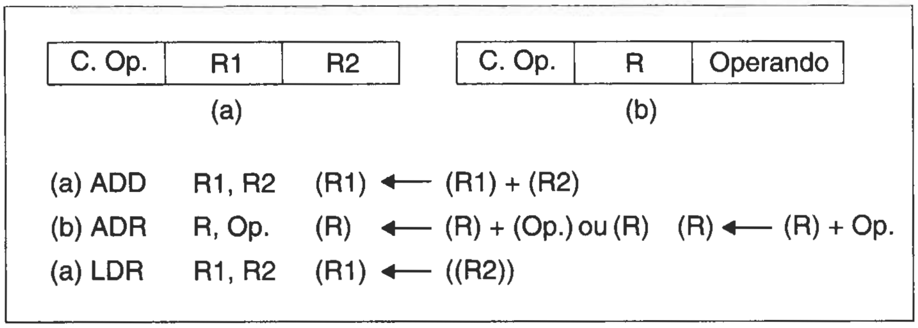
No primeiro caso, o registrador endereçado na instrução contém o dado a ser manipulado. No outro, o registrador referenciado armazena o endereço de uma célula de memória onde se encontra o dado. A instrução conterá, como sempre, o endereço do registrador.

A Fig. 8.8 mostra dois tipos de instruções que empregam o modo de endereçamento por registrador.

Na Fig. 8.8(a) observa-se que a instrução possui dois campos contendo, cada um, o *endereço* de um registrador (um dos 16 possíveis), o qual poderá ter armazenado o dado (direto) ou o endereço da MP, onde estará o dado (indireto).

Na Fig. 8.8(b), a instrução possui dois campos de operando, um dos quais é o endereço de um registrador (que contém o dado) e o outro é o endereço de uma célula da MP, que armazena o dado (direto). Pode-se, ainda, ter uma variação desse formato, em que o campo operando pode conter o próprio valor do dado (imediato).

Embora o modo de endereçamento por registrador seja vantajoso em vários aspectos, tais como rapidez de execução da instrução e economia de espaço de armazenamento das instruções, essas vantagens nem sempre



```
(a) ADD   R1, R2   (R1) ←— (R1) + (R2)
(b) ADR   R, Op.   (R)  ←— (R) + (Op.) ou (R)   (R) ←— (R) + Op.
(a) LDR   R1, R2   (R1) ←— ((R2))
```

Figura 8.8 Exemplo de instruções com modo de endereçamento por registrador.

são aplicáveis. Há certos casos em que não se observa vantagem alguma no referido método, ocorrendo até desperdício de instruções, o que pode vir a constituir uma desvantagem.

No exemplo utilizado, as instruções que efetivamente calculam a equação X = A + B não empregaram modo de endereçamento por registrador e sim o modo direto convencional.

Não é eficaz usar registrador apenas para realizar uma transferência do tipo:

MP → R → UAL

Como podemos verificar, os dados representados pelas variáveis A e B estarão, em cada uma das 100 execuções, sempre na MP (são lidos do meio exterior para a MP) e terão que ser passados para a UAL as 100 vezes. Assim, se utilizássemos algum registrador para armazenar A ou B, o dispositivo serviria apenas para atrasar a execução da instrução com mais um armazenamento, sem se obter qualquer vantagem.

O uso do registrador somente é vantajoso se proporcionar redução de ciclos de memória, o que não era o caso.

Uma outra possível desvantagem do emprego do modo de endereçamento por registrador consiste, em certos casos, na dificuldade em se definir quais dados serão armazenados nos registradores e quais permanecerão na MP (por falta de registradores). Isso acontece devido ao reduzido número dos registradores existentes nas UCPs e à grande quantidade de dados manipulados pelos programas.

8.2.5 Modo Indexado

Freqüentemente, durante a execução de programas, há necessidade de se manipular endereços de acesso a elementos de certos tipos especiais de dados. Esses endereços servem, na realidade, de ponteiros para os referidos elementos.

Por exemplo, o acesso aos elementos de um vetor (array) deve considerar que tais elementos são armazenados seqüencialmente na memória e que sua localização pode ser referenciada por um ponteiro (endereço), que é alterado para indicar o elemento desejado (o índice do elemento identifica individualmente cada um).

Portanto, é importante que haja, no conjunto de instruções de máquina, algumas capazes de realizar essas manipulações de endereços, permitindo uma localização dos dados mais rápida e eficiente.

A maioria dos processadores possui uma ou mais dessas instruções; sua descrição caracteriza um modo de endereçamento denominado *indexado*. Essa denominação advém do fato de que a obtenção do endereço de um dado (elemento de um array) relaciona-se com seu índice.

Nesse tipo de instrução, o endereço do dado é a soma do valor do campo operando (fixo para todos os elementos de um dado array) e de um valor armazenado em um dos registradores da UCP (normalmente denominado *registrador índice*). O valor armazenado nesse registrador varia para o acesso a cada elemento ("aponta" para o elemento desejado).

Na verdade, esse modo de endereçamento é uma evolução das técnicas desenvolvidas desde os primórdio da computação para manipulação dessas estruturas de dados especiais.

Podemos exemplificar essa assertiva apresentando alguns possíveis métodos, evolutivamente usados manipulação de arrays, até atingirmos a eficiência do modo indexado.

Consideremos, por exemplo, a necessidade de, em certo programa, executar-se a seguinte operação sobre três vetores (arrays) de 100 elementos cada:

```
Prog-1      DO I = 1 TO 100
            C(I) = A(I) + B(I)
```

Usando o Modo Direto sem Alterar os Bits que Descrevem as Instruções

Nesse caso, haveria necessidade de se escrever instruções para cada uma das 100 operações de soma a serem efetivamente realizadas pela máquina. Exemplificando com as instruções de um operando adotadas nesse texto, teríamos um programa semelhante ao mostrado na Fig. 8.9.

Evidentemente, essa técnica de programação é ineficiente e trabalhosa, usando o computador apenas como calculadora, já que o programador terá toda a carga de trabalho.

```
LDA A(1)
ADD B(1)
STA C(1)
LDA A(2)
ADD B(2)
STA C(2)
-----------
-----------
LDA A(100)
ADD B(100)
STA C(100)
HLT
```

Figura 8.9 Programa 1, em linguagem Assembly.

Usando o Modo Direto com Alteração Dinâmica do Conteúdo das Instruções

O cálculo da soma dos 100 elementos pode ser obtido por um programa com muito menor quantidade de instruções. Nesse caso, emprega-se o automatismo do computador para realizar a tarefa de executar 100 vezes as operações.

Para tanto, é necessário que se determine, em tempo de execução, o endereço de cada um dos 100 elementos dos vetores. Como esses elementos estão armazenados seqüencialmente na MP, basta existir uma instrução que incremente o valor do campo operando (endereço do dado). Tal instrução executa uma operação aritmética (X = X + 1) sobre um valor binário (X), que é, na realidade, um endereço.

O programa, em Assembly, para resolver o algoritmo definido em Prog-1, seria semelhante ao mostrado na Fig. 8.10.

Programa Assembly	Programa em linguagem de máquina
T LDA A(1)	11A00
1 ADD B(1)	21A64
2 STA C(1)	31AC8
INC T	8103A
INC 1	8103B
INC 2	8103C
DCR N	919FF
JNZ T	D103A
END	F0000

Figura 8.10 Programa 2, em linguagens Assembly e de máquina.

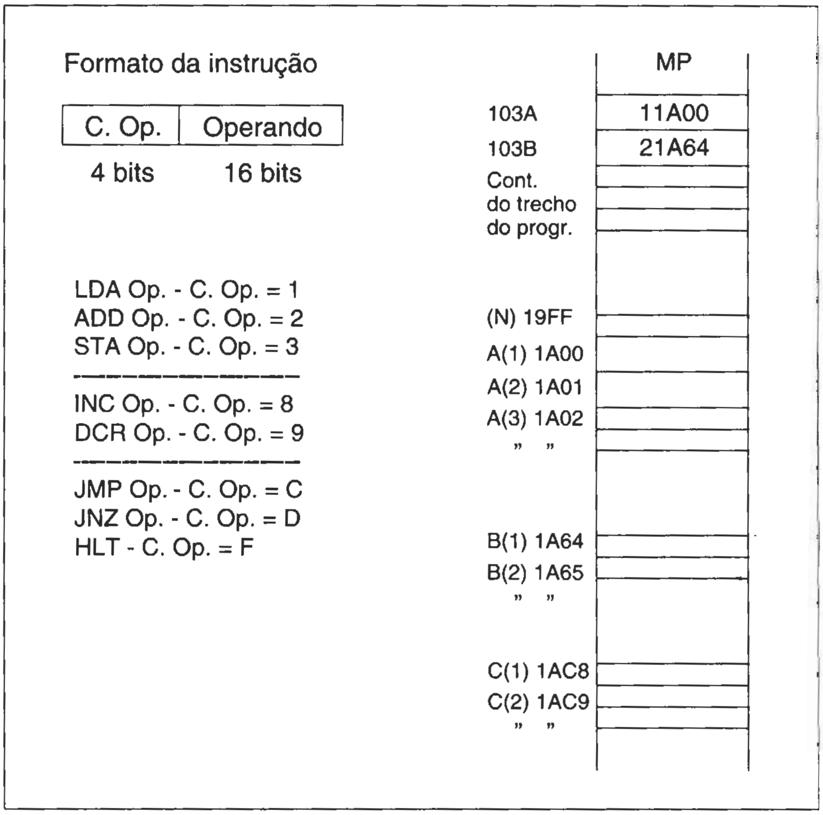


Figura 8.11 Programa e dados de uma soma de vetores no modo direto.

Observe a Fig. 8.11, na qual é apresentada uma MP com 64K células (cada célula é identificada por um endereço com 4 algarismos hexadecimais), todas com capacidade de armazenar valores com 20 bits (5 algarismos hexadecimais) e instruções (de um operando), do tamanho da célula e da palavra.

Nessa memória foram armazenados os elementos do vetor A (a partir do endereço hexadecimal 1A00), do vetor B (a partir do endereço hexadecimal 1A64) e, a partir do endereço 1AC8, deverão ser armazenados os elementos do vetor C (resultado da soma). O trecho do programa (Fig. 8.10) começa a partir do endereço hexadecimal 103A (o conteúdo da célula tem armazenada a instrução cujo valor em hexadecimal é 11A00).

A primeira execução da instrução INC, código de operação igual a 8 (utilizando o modo de endereçamento imediato), acarreta a alteração do valor do campo operando das instruções LDA, ADD e STA, respectivamente, para: 1A01 (endereço do segundo elemento do vetor A, identificado por A(2)), 1A65 (endereço do elemento B(2) do vetor B) e 1AC9 (endereço do elemento C(2) do vetor C). Na passagem seguinte do loop, os valores seriam novamente alterados (somando 1 ao valor atual) para, respectivamente: 1A02, 1A66 e 1ACA, endereços dos elementos A(3), B(3) e C(3), dos vetores A, B e C e assim sucessivamente, até alcançar-se o endereço dos últimos elementos: A(100), B(100) e C(100).

O método é adequado e vantajoso, na medida em que são elaboradas apenas *nove* instruções (é claro que há outras maneiras de se fazer o mesmo programa), e o trabalho de execução fica por conta da máquina, não do programador.

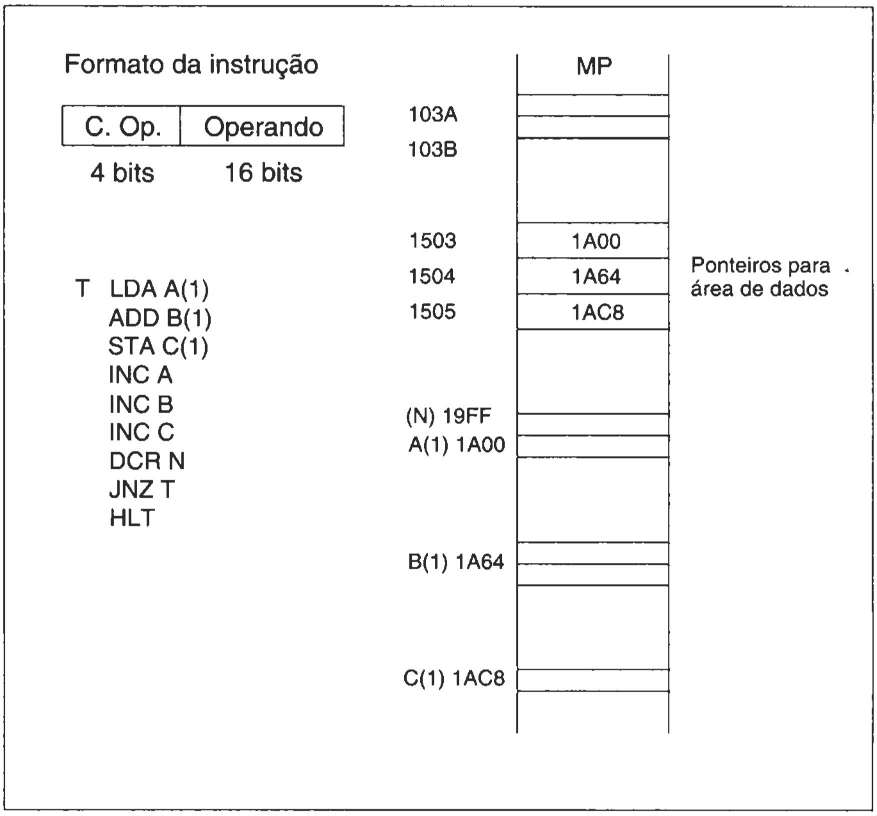
A desvantagem deste método, porém, reside no fato de que o valor de uma instrução (o campo operando) é alterado durante a execução do programa. Caso houvesse alguma interrupção anormal no meio da execução, seria preciso reinicializar os valores iniciais, bem como a reinicialização dos valores teria que ser realizada em toda nova execução do programa, pois ele termina com os valores de endereço de A(100), B(100) e C(100).

Usando o Modo Indireto

Essa é uma das aplicações do modo indireto, pois o endereço de cada elemento do vetor estará armazenado na MP, em uma célula cujo endereço consta no campo operando da instrução.

O processo de obtenção do endereço de cada elemento, durante a execução do programa, consiste na alteração (adicionando-se um valor que aponte para o novo elemento) do conteúdo da posição de memória endereçada, no modo indireto, pelo campo operando da instrução. Assim, não há modificação das instruções em tempo de execução (um dos problemas do método anterior).

A Fig. 8.12 apresenta um exemplo dessa técnica. Os elementos de cada vetor estão armazenados nos mesmos endereços da figura anterior. As instruções LDA, ADD e STA empregam o modo indireto, enquanto INC usa o modo direto (para incrementar os endereços dos elementos dos vetores).



```
T  LDA A(1)
   ADD B(1)
   STA C(1)
   INC A
   INC B
   INC C
   DCR N
   JNZ T
   HLT
```

Figura 8.12 Soma de vetores (programas e dados) usando o modo indireto.

O programa é basicamente semelhante ao do exemplo anterior (modo direto), exceto que a primeira execução das instruções INC somaria 1 aos valores 1A00, 1A64 e 1AC8 (respectivamente, conteúdo das células de endereços 1503, 1504 e 1505). Isso permitiria apontar para os elementos A(2), B(2) e C(2).

Outra diferença fundamental em relação ao método anterior é que, nesse caso, os valores de endereços dos elementos não fazem parte das instruções; estão armazenados na MP e, portanto, as instruções não são alteradas durante a execução do programa.

Essa é uma técnica mais limpa, e eficiente, em termos de programação, embora haja um gasto maior de tempo na execução do programa, devido à maior quantidade de acessos do modo indireto.

Usando o Modo Indexado

Nesse modo, o endereço de cada elemento do vetor é obtido pela soma (a soma é efetuada antes da colocação do valor do endereço de acesso no REM) do valor do campo operando da instrução com o valor armazenado em um dos registradores da UCP (escolhido como registrador-índice). O resultado da soma é o endereço efetivo do dado, o qual será, então, transferido para o REM. A escolha de qual registrador será utilizado como registrador-índice depende da linguagem que se estiver empregando: caso a linguagem seja Assembly, é responsabilidade do programador administrar o uso dos registradores da UCP; se, por outro lado, for empregada uma linguagem de alto nível, a escolha de uso dos registradores fica por conta do programa compilador.

Para utilizar esse modo, é necessário haver instruções que manipulem valores em registradores, tais como:

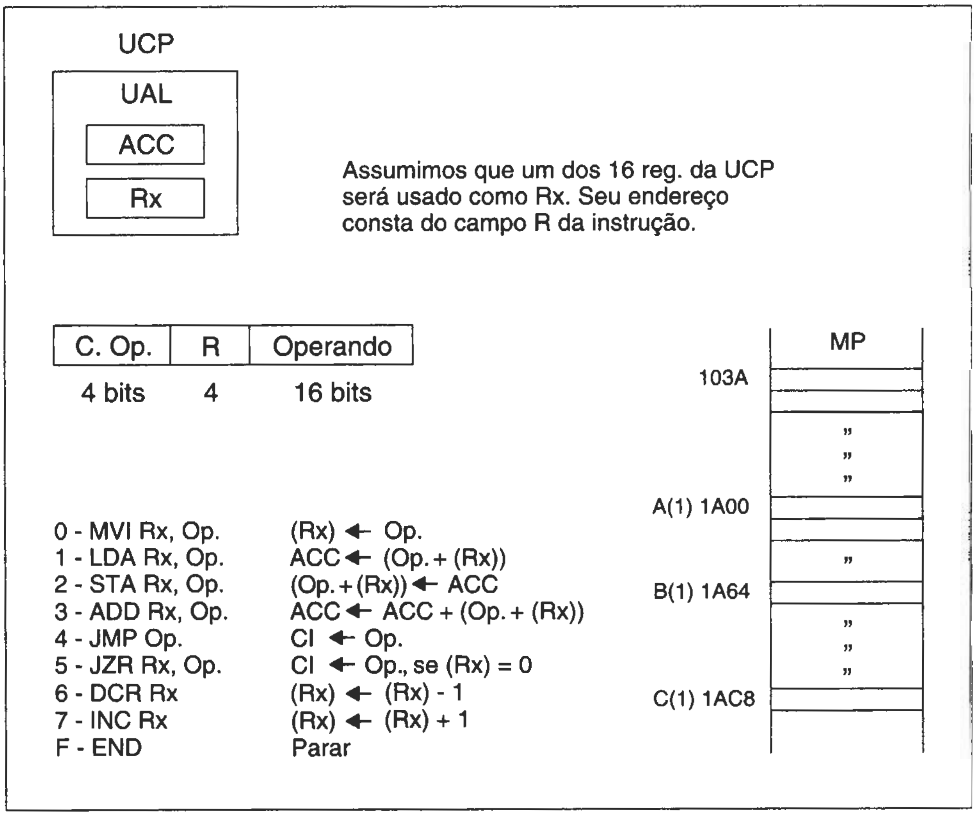
- carregar um valor no registrador (armazenar o índice);
- somar um dado valor ao existente no registrador;
- desviar para outra instrução, se o valor armazenado no registrador for igual a zero (permite sair de um "looping" de execução), e assim por diante.

A grande vantagem da técnica reside na rapidez de execução das instruções de acesso aos dados, visto que a alteração do endereço dos elementos é realizada na própria UCP.

Muitos computadores modernos possuem instruções no modo indexado, cujo exemplo de emprego é apresentado nas Figs. 8.13 e 8.14 (ainda a mesma soma dos vetores A, B e C, dos métodos anteriores — Prog-l).

```
      MVI (R4), 1
      MVI (R2), 100
T     LDA (R4), 19FF
      ADD (R4), 1A63
      STA (R4), 1AC7
      INC (R4)
      DCR (R2)
      JZR (R2), T
      HLT
```

Figura 8.13 Programa 3, em linguagem Assembly.



Assumimos que um dos 16 reg. da UCP será usado como Rx. Seu endereço consta do campo R da instrução.

C. Op.	R	Operando
4 bits	4	16 bits

0 - MVI Rx, Op.	(Rx) ← Op.
1 - LDA Rx, Op.	ACC ← (Op. + (Rx))
2 - STA Rx, Op.	(Op. + (Rx)) ← ACC
3 - ADD Rx, Op.	ACC ← ACC + (Op. + (Rx))
4 - JMP Op.	CI ← Op.
5 - JZR Rx, Op.	CI ← Op., se (Rx) = 0
6 - DCR Rx	(Rx) ← (Rx) - 1
7 - INC Rx	(Rx) ← (Rx) + 1
F - END	Parar

Figura 8.14 Exemplo de emprego do modo indexado.

Em primeiro lugar, move-se (MVI) o valor inicial do índice para o registrador escolhido como Rx (foi escolhido como exemplo o registrador R4) e, assim, teremos a instrução, em linguagem de máquina:

$0\,4\,0\,0\,0\,1_{16}$

Em seguida, inicia-se o trecho repetitivo do programa, executando o teste de verificação da saída do looping (quando se atingir a soma do centésimo elemento de cada um dos vetores, A e B).

A soma propriamente dita é obtida pela repetição de:

LDA Rx, Op.	Começa carregando A(1)
ADD Rx, Op.	Somar A e B
STA Rx, Op.	Armazenar resultado em C

Em primeiro lugar, o endereço de A(1), (1A00), é indicado na instrução pela soma do campo Op. (no caso presente seria, por exemplo, o valor hexadecimal 19FF) com o conteúdo de Rx (no caso presente, o conteúdo de R4, que deverá ser 0001). O resultado dessa soma, 1A00, será, então, transferido para o REM, iniciando o ciclo de leitura do elemento A(1) do vetor A.

A obtenção do valor do elemento de B(1) do vetor B, cujo endereço em hexadecimal é 1A64, também será realizada pela soma do valor 0001 (armazenado em R4) com o valor 1A63 (valor constante do campo Op. — sempre o mesmo em todas as instruções que manipulem elementos do vetor B), e assim também será efetuado para o cálculo do endereço dos elementos do vetor C.

O programa prossegue, com o incremento do valor armazenado em R4 (Rx), de modo a permitir a soma do elemento A(2) com o elemento B(2). A instrução seria INC Rx. Nesse caso, o valor de R4 passaria de 0001 para 0002, o que, somado ao valor do campo Op. apontaria para o segundo elemento de cada vetor.

Em seguida, retorna-se ao ponto inicial, para testar se o loop já foi executado 100 vezes. Caso negativo, a soma é reiniciada: o valor 19FF será somado ao valor 0002 (para endereço de A(2)), bem como 1A63 e 1AC7 com 0002, respectivamente, para endereços de B(2) e C(2). E assim, sucessivamente, até o final.

8.2.6 Modo Base Mais Deslocamento

Este modo de endereçamento tem características semelhantes ao modo indexado, visto que o endereço de acesso a uma célula de memória se obtém pela soma de dois valores, um inserido no campo apropriado da instrução (normalmente denominado campo deslocamento — *displacement*) e o outro valor inserido em um determinado registrador, denominado registrador-base ou registrador de segmento.

A diferença entre eles está na aplicação e no propósito do método e, por conseguinte, na forma de implementá-lo. Neste caso, o valor a se manter fixo é o do registrador-base/segmento, variando o conteúdo do campo deslocamento em cada instrução, diferentemente do modo indexado, em que o conteúdo do registrador é que se altera.

Os processadores da família Intel x86 possuem alguns registradores projetados especificamente com a finalidade de servir como registrador de segmento, como os 6 registradores de 16 bits dos processadores Pentium.

Este modo de endereçamento acarreta uma redução do tamanho das instruções (e, com isso, economiza memória), bem como facilita o processo de relocação dinâmica de programas.

A sua escolha decorre de dois fatores:

a) durante a execução de uma grande quantidade de programas, as referências a células de memória, onde se localizam os operandos, normalmente são seqüenciais, ocorrendo poucos acessos a outras instruções fora de ordem (exceto desvios); e

b) a maioria dos programas ocupa um pequeno espaço da MP disponível.

Dessa forma, em vez de ser necessário, em cada instrução, que o campo operando tenha um tamanho correspondente à capacidade total de endereçamento da MP, basta que o endereço desejado seja obtido pela soma de um valor existente em um dos registradores da UCP com um valor contido na instrução.

Por isso o método é chamado de *base mais deslocamento*, consistindo, então, na utilização de dois campos na instrução (que substituem o campo operando): um, com o endereço de um registrador (chamado de *base* ou *segmento*), e outro, com um valor denominado *deslocamento* (porque contém um valor relativo — que se desloca em relação — à primeira instrução).

Consideremos, por exemplo, o caso de processadores, nos quais as instruções possuem campo de endereço de registrador-base com 4 bits (estipulamos no exemplo que o processador possui 16 registradores e, portanto, será necessário se definir 16 endereços, um para cada registrador) e campo deslocamento, com 12 bits. E que o processador pode endereçar até 16M células (cada endereço linear de memória deverá ter 24 bits).

Nesse caso, podem-se endereçar áreas de 4096 bytes (4K) com um valor armazenado no registrador-base, gastando-se apenas 16 bits (4 + 12), ao contrário dos 24 bits necessários para endereçar diretamente todas as células da MP daquele computador (capacidade máxima da MP igual a 16 Mbytes). Economizam-se, assim, 8 bits em cada instrução.

A Fig. 8.15 apresenta um exemplo desse modo de endereçamento, usando quatro registradores-base e 12 bits para o campo deslocamento. Pode-se observar então que, em um dado instante, há quatro áreas de endereçamento, cada uma correspondente ao valor armazenado em cada registrador-base.

Da descrição dos modos *indexado* e *base mais deslocamento*, podemos observar que o processo de cálculo do efetivo endereço de acesso é idêntico em ambos os modos. A diferença está no conceito de cada um, não na sua implementação.

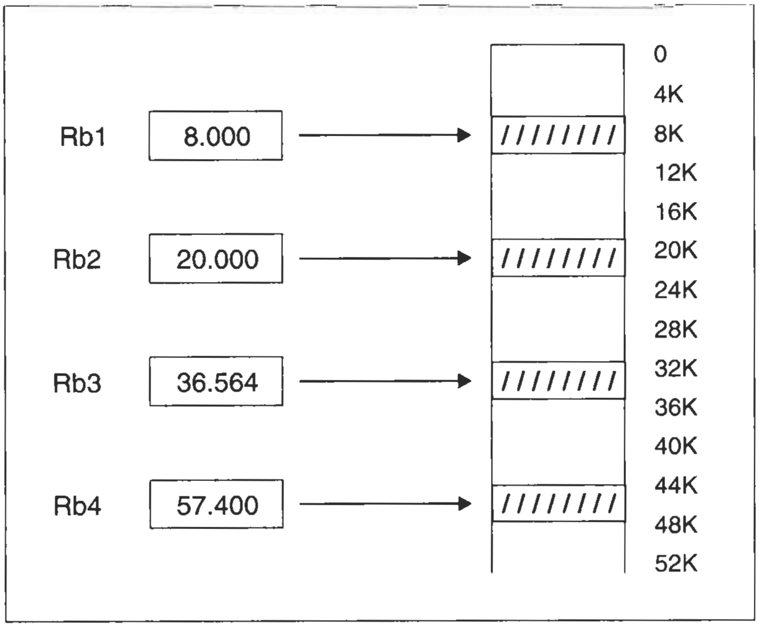


Figura 8.15 Exemplo de uso do modo base mais deslocamento.

A indexação é empregada quando se deseja acessar diferentes dados, com alteração do endereço, por incremento (ou decremento) do valor do registrador-índice. Quando a modificação de endereço é realizada para relocação de programa, basta uma única alteração do conteúdo do registrador-base (no modo base mais deslocamento).

Sobre o que realmente acontece na máquina, podemos observar:

a) vários dados são acessados com diversos valores de registrador-índice e um único valor no campo operando;

b) vários dados são acessados com um único valor de registrador-base e valores diferentes no campo deslocamento da instrução.

EXERCÍCIOS

1) Cite uma possível vantagem do emprego de instruções com menor quantidade de operandos.

2) Crie um conjunto de instruções de dois operandos, definidas em linguagem Assembly, necessárias para a realização de operações aritméticas e elabore programas para cálculo das seguintes equações:

   a) $X = A + (B * (C - A) + (D - E/B) * D)$

   b) $Y = (A + B * (C - D * (E/(B - F)) + B) * E)$

3) Considere as instruções Assembly de um operando utilizadas neste livro e escreva programas para resolver as equações apresentadas no exercício anterior.

4) Faça o mesmo para instruções de dois operandos.

5) Cite uma aplicação em programa para o modo de endereçamento imediato. Indique uma desvantagem desse modo.

6) Faça o mesmo para o modo de endereçamento direto.

7) Analise os modos de endereçamento direto e direto por registrador, estabelecendo diferenças de desempenho, vantagens e desvantagens de cada um.

8) Considere as instruções definidas a seguir (de um operando):

LDA Op.	ACC ← (Op.)	STA Op.	(Op.) ← ACC
ADD Op.	ACC ← ACC + (Op.)	SUB Op.	ACC ← ACC − (Op.)
MPY Op.	ACC ← ACC * (Op.)	DIV Op.	ACC ← ACC/(Op.)

   Deduza a equação matemática cuja solução resultou no seguinte programa, criado com estas instruções:

```
LDA   A
ADD   C
STA   X
LDA   B
MPY   D
SUB   E
STA   Y
LDA   X
ADD   Y
DIV   F
STA   X
```

9) Qual é o objetivo do emprego do modo de endereçamento base mais deslocamento? Qual é a diferença de implementação entre esse modo e o modo indexado?

10) Em um determinado processador, há instruções que usam o modo de endereçamento base mais deslocamento, cada uma possuindo um tamanho de X bits. Desses X bits, **a** bits identificam o código da operação; **b** bits especificam o endereço do registrador usado como base; **c** bits são empregados para o campo deslocamento. Considerando que a barra de endereços possui **y** bits, que fração da MP pode ser endereçada sem que sejam alterados os conteúdos dos registradores-base existentes nesse processador?

11) Considere um processador que possua as seguintes características:

- um registrador de 8 bits;
- um registrador de 16 bits;
- uma barra de dados de 8 bits;
- uma barra de endereços de 16 bits.

Defina instruções que permitam ao processador carregar em um registrador o conteúdo do endereço dado, adicionar a um registrador um valor especificado e carregar no registrador A o conteúdo da posição de memória apontada pelo registrador B. Descreva cada instrução e caracterize o tipo de endereçamento que ela utiliza. Com essas instruções faça um programa que permita carregar no registrador A o elemento de ordem 1F de uma tabela que começa na posição de memória de endereço 013D e que gasta 8 bits de memória para cada elemento.

12) Considere um computador com UCP constituída de um RI com 24 bits, CI e REM de 12 bits, UAL, UC e vários registradores de emprego geral. Esse computador possui um conjunto de 256 instruções de formato único, mostrado a seguir, e modos de endereçamento: *direto, indireto* e *por registrador*.

C. Op.	R1	R2	Operando

a) Quantos registradores de emprego geral podem ser endereçados nesse processador?

b) Supondo duas instruções A e B, onde a instrução A acessa a MP no modo indireto e a instrução B acessa a MP no modo por registrador (modalidade indireta), qual delas executa seu ciclo de instrução mais rápido? Por quê?

9

Execução de Programas

9.1 INTRODUÇÃO

Conforme já foi mencionado nos capítulos anteriores, um computador, para realizar uma tarefa específica, como, por exemplo, somar 10 números em seqüência, precisa ser instruído, passo a passo, para efetivamente realizar a tarefa. Necessita, pois, que seja projetado com a capacidade de realizar (interpretar e executar) um determinado conjunto de operações, cada uma sendo constituída de uma instrução específica (instrução de máquina). O conjunto de instruções ou comandos organizados em uma certa seqüência, para obter o resultado da soma dos 10 números, compõe o que denominamos programa de computador (vamos denominar apenas *programa*, já que neste livro não estamos tratando de outros tipos de programas).

No item 6.4 descrevemos o formato das instruções de máquina normalmente encontradas nas UCP e também o procedimento, passo a passo, para a execução de uma instrução de máquina (ciclo de instrução).

Atualmente, é raro escrever-se um programa diretamente na linguagem da máquina em virtude da enorme dificuldade de se organizar as instruções sem erros (quanto maior o programa, maior é a possibilidade de erros, enganos ou omissões) e dos problemas de entendimento e manutenção do programa tanto por parte de outros programadores como até mesmo por quem criou o referido programa.

Embora os métodos encontrados para solucionar esses problemas sejam característicos da tecnologia de software, não fazendo, portanto, parte do escopo deste livro, consideramos conveniente incluir uma breve descrição dos métodos e técnicas existentes, de modo a facilitar o trabalho de programação, essencialmente no que se refere ao uso de linguagens que não sejam de máquina e aos procedimentos para conversão dos programas escritos nessas linguagens para linguagem de máquina.

Vamos apresentar, então, um resumo das etapas que definem o processo de execução de um programa escrito em uma linguagem qualquer, de nível acima da linguagem de máquina, descrevendo cada uma dessas etapas, para posteriormente consolidar o entendimento das explicações por meio de exemplo de execução completa de programas.

9.2 LINGUAGENS DE PROGRAMAÇÃO

Uma *linguagem de programação* é uma linguagem criada para instruir um computador a realizar suas tarefas. Um programa completo, escrito em uma linguagem de programação, é freqüentemente denominado *código*. Deste modo, codificar um algoritmo significa converter suas declarações em um comando ou instrução específico de uma certa linguagem de programação.

O tipo mais primitivo de linguagem de programação é a linguagem que o computador entende diretamente, isto é, as instruções que podem ser diretamente executadas pelo hardware, isto é, pela UCP. É a *linguagem de máquina* (ver itens 6.3 e 6.4), que foi utilizada pela primeira geração de programadores.

Conforme já descrevemos anteriormente, uma instrução de máquina é um conjunto de bits, dividido em subconjuntos ou campos, com funções determinadas: um subconjunto (um campo da instrução) estabelece o código de operação e o outro define a localização do(s) dado(s). Um programa em linguagem de máquina é, em conseqüência, uma longa seqüência de algarismos binários, alguns dos quais representam instruções e outros, os dados a serem manipulados pelas instruções. A Fig. 9.1 mostra um exemplo de programa em linguagem de máquina, na sua forma binária pura (Fig. 9.1(a)), e uma representação mais compacta e um pouco mais simples, em linguagem hexadecimal (Fig. 9.1(b)).

```
0010  0100  1001  0001                2 4 9 1
0100  0100  1001  1111                4 4 9 F
0100  0100  1001  0011                4 4 9 3
0001  0100  1001  0010                1 4 9 2
1000  0100  1001  1000                8 4 9 8
1110  0100  1001  1001                E 4 9 9
0011  0100  1001  0101                3 4 9 5
0100  0100  1001  1110                4 4 9 E
1111  0100  1001  1010                F 4 9 A
0000  0000  0000  0000                0 0 0 0
```

(a) Programa em linguagem binária (b) Programa em hexadecimal

Figura 9.1 Programa em linguagens binária e hexadecimal.

Para criar um programa em linguagem de máquina, o programador deve conhecer todas as instruções disponíveis para aquela máquina e seus respectivos códigos de operação e formatos, assim como os registradores da UCP disponíveis e os endereços das células de memória onde serão armazenadas as instruções e os dados. Um programa real, em linguagem de máquina, pode conter milhares de instruções, o que é uma tarefa extremamente tediosa e difícil, pelos detalhes que precisam ser observados pelo programador. É, portanto, caro de realizar pelo custo da mão-de-obra envolvida.

Para tentar minimizar esta ineficiência (dificuldade de entendimento do significado dos números, todos semelhantes), foi desenvolvida, ainda para a primeira geração de computadores, uma linguagem que representasse as instruções por símbolos e não por números. Esta linguagem simbólica foi denominada Linguagem de Montagem (Assembly Language), cujos conceitos básicos foram apresentados no item 6.5.

Códigos de operação, como 0101, são mais fáceis de serem lembrados se representados como ADD (somar) do que pelo número 0101. Além disso, o programador, ao escrever um programa em linguagem de montagem (Assembly), não precisa mais guardar os endereços reais de memória onde dados e instruções estarão armazenados. Ele pode usar símbolos (caracteres alfabéticos ou alfanuméricos para indicar endereços ou dados), como MATRÍCULA, NOME, SALÁRIO, para indicar os dados que são usados em um programa.

Uma instrução em linguagem de máquina do tipo:

1110 0100 1001 1001 ou E 4 9 9

pode ser mais facilmente entendida se representada na forma simbólica:

ADD SALARIO

A Fig. 9.2 apresenta o programa exemplificado na Fig. 9.1, porém, neste caso, escrito em linguagem de montagem (Assembly).

Para se usar linguagem de montagem em um computador, é necessário que haja um meio de converter os símbolos alfabéticos utilizados no programa em código de máquina, de modo a poder ser compreendido pela UCP (pelo processador). O processo de conversão (também denominado tradução) é chamado de *montagem* e é realizado por um programa denominado *Montador* (Assembler). O montador lê cada instrução em linguagem de montagem e cria uma instrução equivalente em linguagem de máquina.

```
ORG          ORIGEM
LDA          SALARIO - 1
ADD          SALARIO - 2
ADD          SALARIO - 3
SUB          ENCARGO
STA          TOTAL
HLT
DAD          SALARIO - 1
DAD          SALARIO - 2
DAD          SALARIO - 3
```

Figura 9.2 Programa da Fig. 9.1 em linguagem de montagem.

Embora escrever um programa em linguagem de montagem ainda seja uma tarefa árdua, tediosa e complexa (atualmente poucos programas são escritos nessa linguagem, em comparação com o que ocorria há alguns anos), ainda é bem mais atraente do que escrever o mesmo programa entre essas duas linguagens.

Um passo mais significativo no sentido de criar uma linguagem de comunicação com o computador, mais simples e com menos instruções do que a linguagem de montagem, foi o desenvolvimento de linguagens que refletissem mais os procedimentos utilizados na solução de um problema, sem preocupação com o tipo de UCP ou de memória onde o programa será executado. Tais linguagens, por serem estruturadas de acordo com a compreensão e a intenção do programador, são usualmente denominadas *linguagens de alto nível*, nível afastado da máquina.

Tabela 9.1 Exemplos de Linguagens de Programação de Alto Nível

Linguagem	Data	Observações
FORTRAN	1957	FORmula TRANslation — primeira linguagem de alto nível. Desenvolvida para realização de cálculos numéricos.
ALGOL	1958	ALGOrithm Language — linguagem desenvolvida para uso em pesquisa e desenvolvimento, possuindo uma estrutura algorítmica.
COBOL	1959	COmmon Business Oriented Language — primeira linguagem desenvolvida para fins comerciais.
LISP	1960	Linguagem para manipulação de símbolos e listas.
PL/I	1964	Linguagem desenvolvida com o propósito de servir para emprego geral (comercial e científico). Fora de uso.
BASIC	1964	Desenvolvida em Universidade, tornou-se conhecida quando do lançamento do IBM-PC, que veio com um interpretador da linguagem, escrito por Bill Gates e Paul Allen.
PASCAL	1968	Primeira linguagem estruturada — designação em homenagem ao matemático francês Blaise Pascal que, em 1642, foi o primeiro a planejar e construir uma máquina de calcular.
C	1967	Linguagem para programação de sistemas operacionais e compiladores.
ADA	1980	Desenvolvida para o Departamento de Defesa dos EUA.
DELPHI	1994	Baseada na linguagem Object Pascal, uma versão do Pascal orientada a objetos.
JAVA	1996	Desenvolvida pela Sun, sendo independente da plataforma onde é executada. Muito usada para sistemas Web.

Uma *linguagem de alto nível*, ou orientada ao problema, permite que o programador especifique a realização de ações do computador com muito menos instruções (chamaremos de *comando* as instruções referentes às linguagens de alto nível).

Desde o aparecimento de linguagens como FORTRAN e ALGOL, na década de 1950, dezenas de outras linguagens de alto nível foram desenvolvidas, seja para uso geral seja para resolver tipos mais específicos de problemas. A Tabela 9.1 apresenta algumas das mais conhecidas linguagens de programação de alto nível, indicando-se a época de seu surgimento no mercado.

A Fig. 9.3 apresenta partes de um programa em COBOL (Fig. 9.3(a)) e de um programa em C (Fig. 9.3 (b)). Conquanto a simples observação de cada código possa induzir a impressão de uma grande diferença entre os dois códigos, na realidade eles têm o mesmo propósito e produzem o mesmo resultado. Da mesma forma que os programas com instruções de máquina, os programas da Fig. 9.3, ou qualquer outro programa em linguagem de alto nível, também requerem uma conversão para instruções de máquinas, processo denominado compilação (ver item 9.3.2). Normalmente, a conversão de um simples comando em linguagem COBOL ou C, como os da Fig. 9.3, redunda em dezenas de instruções de máquina, diferentemente de um programa em linguagem de montagem, no qual cada instrução implica essencialmente uma única instrução de máquina.

9.3 MONTAGEM E COMPILAÇÃO

No item anterior verificamos que programas de computador não são escritos na linguagem que a máquina entende, mas sim em outras formas simbólicas de representar as instruções que o programador deseja que se-

```
DETAIL PARAGRAPH.
        READ CARD-FILE AT END GO TO END-PARAGRAPH.
        MOVE CORRESPONDING CARD-IN TO LINE-OUT.
        ADD CURRENT-MONTH-SALES IN CARD-IN, YEAR-TO-DATE-SALES IN
                CARD-IN GIVING TOTAL-SALES-OUT IN LINE-OUT.
                WRITE LINE-OUT BEFORE ADVANCING 2 LINES AT EOP
                PERFORM HEADER-PARAGRAPH
```

(a) Trecho de programa em COBOL.

```
detail_procedure ( )
{
        int file_status, current_line, id_um;
        char *name;
        float current_sales, ytd_sales, total_sales;

        current_line=64;
        file_status=fscanf (card_file, "%d%s%f%f", &id_num, &name, &current_sales,
                &ytd_sales);
        while (file_status  !=EOF) {
        if (current_line > 63) {
                header_procedure ( );
                current_line=3;
        }
        file_status=fscanf (card_file, "%d%s%f%f", &id_num, &name, &current_sales,
                &ytd_sales);
        total_sales = current_sales + ytd _sales;
        printf ("%4d%30s%6.2f%7.2f\n\n", id_num, name, current_sales,
                ytd_sales, total_sales);
        current_line ++; current_line ++;
        }
end_procedure ( );
```

(b) Trecho de programa em C.

Figura 9.3 Exemplo de trechos de programas em COBOL e em C.

jam realizadas. No entanto, verificamos que as máquinas continuam entendendo somente em binário e, por isso, sempre há necessidade de conversão ou tradução de um programa em linguagem simbólica para outro, equivalente, em linguagem numérica binária.

9.3.1 Montagem

A tradução mais rápida e simples que existe denomina-se *montagem* e é realizada por um programa denominado *montador* (Assembler). Como o nome já explica, a montagem é realizada para traduzir um programa em linguagem de montagem para seu equivalente em linguagem binária, executável.

A Fig. 9.4 mostra o fluxo básico de uma montagem. Nele, o programa escrito em linguagem de montagem, chamado de código-fonte, é examinado instrução por instrução e, em seguida, é convertido para um outro programa em linguagem binária de máquina, denominado *código-objeto*. A Fig. 9.5 mostra um exemplo de um programa em linguagem de montagem e o seu respectivo código em linguagem de máquina.

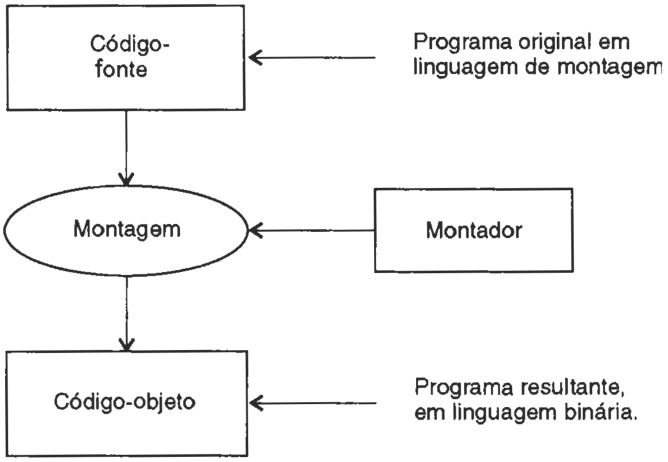

Código-fonte
Programa original em linguagem de montagem
Montagem
Montador
Código-objeto
Programa resultante, em linguagem binária.

Figura 9.4 Fluxo do processo de montagem.

Basicamente, as funções de um montador são:

- Substituir códigos de operação simbólicos por valores numéricos, isto é, substituir LOAD, STR, ADD, MOV, etc. por 00101101, 100010, 00001110 e 11111100 (os valores binários mostrados para os códigos de operação são meros exemplos, sem preocupação com qualquer processador real).
- Substituir nomes simbólicos de endereços por valores numéricos dos endereços, isto é, substituir o SOMA de ADD SOMA por 1011100011010 (também se trata de um valor meramente para exemplo).
- Reservar espaço de memória para o armazenamento das instruções e dados.
- Converter valores de constantes para código binário, como, por exemplo, MOV AL, 2214, que converte o valor 2216 para o valor 001000102.
- Examinar a correção de cada instrução (LDA pode ser uma palavra-chave correta, mas LDB não, acarretando um erro. O montador não pode gerar um código de operação de uma instrução que não existe).

Em linhas gerais, estas funções podem ser resumidas no exemplo descrito a seguir. Consideremos o funcionamento de um montador de módulos ou de dois passos e a existência de um determinado sistema de computação que possua no seu conjunto de instruções algumas *instruções de um operando* relacionadas na Fig. 9.6 e que são bastante semelhantes às instruções definidas e exemplificadas no Cap. 6.

Na Fig. 9.6(a) é apresentado o subconjunto de instruções utilizadas para nosso exemplo, enquanto a Fig. 9. (b) relaciona um programa na linguagem de montagem definida.

```
0000              ADDS      PROC      NEAR

0000   03  C3               ADD       Ax, Bx
0002   03  C1               ADD       Ax, Cx
0004   03  C2               ADD       Ax, Dx
0006   C3                   RET
0007              ADDS      ENDP
```

Figura 9.5 Programa em linguagem de montagem.

C. Op.	Descrição
HLT	Parar
INC	ACC ← ACC + 1
DCR	ACC ← ACC - 1
LDA Op.	ACC ← M (Op.)
STR Op.	M (Op.) ← ACC
ADD Op.	ACC ← ACC + M (Op.)
SUB Op.	ACC ← ACC - M (Op.)
JMP Op.	CI ← Op.
JP Op.	Se ACC > 0, então: CI ← Op.
JZ Op.	Se ACC = 0, então: CI ← Op.

(a) Subconjunto de instruções

```
Início :  ORG   ZERO          ; Origem do programa. Endereço relativo ø.
          LDA   CONTADOR      ; Carregar valor do contador no ACC.
          JZ    FIM           ; Se contador = 0, então PARAR ( desvia para FIM ).
          LDA   Parcela 1   ⎫
          ADD   Parcela 2   ⎬ ; Realizar operação aritmética com dados.
          STR   Resultado   ⎭
          LDA   Contador      ; Ler valor do contador para ACC.
          DCR   Zero          ; Substituir 1 do contador.
          JMP   Início        ; Voltar para início do loop.
Fim    :  HLT                 ; Parar.
          DAD   Parcela 1
          DAD   Parcela 2
          DAD   Resultado
          DAD   Contador
```

(b) Programa em linguagem de montagem

Figura 9.6 Exemplo de programa em linguagem de montagem.

Da verificação do programa observa-se que há dois tipos básicos de símbolos: de *códigos de operação* (LDA, STR, JZ etc.) e de *endereços de dados ou de instruções* (INÍCIO, FIM, CONTADOR, PARCELA 1, etc.).

Em um montador de dois passos, o programa é examinado, instrução por instrução, duas vezes. Na primeira vez (primeiro passo), o montador verifica a correção da instrução, ou seja, se ela está corretamente escrita (se é LDA e não LDB, como já exemplificamos) e se possui os campos definidos na estrutura da linguagem de montagem (ver item 6.5). Se encontrar incorreção, o montador registra, de modo a poder relacionar os erros no final da verificação, interrompendo o processo (se o montador encontra erros durante esta fase, ele não prossegue).

Se o código-fonte da instrução estiver correto, ele inicia a descrição de uma tabela, denominada *tabela de símbolos*, em que cada entrada corresponde a um símbolo. Em geral, temos uma tabela de símbolos de códigos de operação e uma tabela de símbolos de endereços.

A partir da pseudo-instrução ORG (ORG não é uma instrução de máquina, mas um símbolo de referência de início do programa, utilizado por muitos montadores como ponto de partida de contagem de endereços), o montador vai criando as entradas da tabela de código de operação, uma para LDA, outra para JZ, outra para ADD, STR, DCR, JMP e HLT. A cada entrada o montador atribui o valor binário do código de operação, além de outros dados pertinentes, como o tamanho da instrução, seus operandos, etc., bem como a posição relativa à instrução inicial do programa.

O montador também cria as entradas na tabela de símbolos de endereços, atribuindo uma entrada a cada símbolo, como CONTADOR, PARCELA 1, etc. Mas o montador não pode concluir a entrada, pois nesse ponto ele não sabe o endereço de memória onde o dado representado simbolicamente por PARCELA 1, por exemplo, será armazenado, nem poderá concluir qualquer outra. Se está examinando a instrução JZ FIM, ele somente sabe o endereço relativo de JZ (é a 2.ª instrução, depois de ORG), mas não sabe que posição relativa corresponde a FIM, o que somente poderá ser definido no segundo passo.

No segundo passo, então, o montador realiza a criação do código-objeto, completando todas as entradas das tabelas. Para tanto, ele passa novamente por cada instrução e a localizará na tabela correspondente. Esta tarefa requer procedimentos otimizados de busca nas tabelas para que o processo de montagem seja rápido.

Neste ponto, então, como ele sabe agora qual a posição relativa de FIM, a entrada de JZ FIM pode ser completada e todas as demais entradas das tabelas também. Finalmente, todo o código de máquina fica completo.

A implementação de uma linguagem de montagem pode ser realizada em um sistema de computação optando-se por um dos dois tipos de montadores:

- montador do tipo carrega e monta ou de 1 passo; e
- montador do tipo de módulos ou de 2 passos.

O segundo tipo, descrito no exemplo anterior, é o método de montagem mais empregado atualmente, visto que, apesar de consumir um certo tempo devido ao fato de que o montador precisa examinar duas vezes todo o programa, ele produz um código de máquina direto no final da montagem, sendo portanto conceitualmente mais simples. Por essa razão, ele permite que possam ser criados vários programas independentes, sendo todos finalmente interligados (ligação ou linkedição) para constituir um único programa executável.

Já o *montador de um passo* (carrega e monta) não tem a mesma clareza de execução do tipo anterior, embora conceitualmente pretenda ser mais rápido. O objetivo deste montador é completar a tarefa em um único passo. Para tanto, durante a avaliação de cada instrução ele vai criando entradas em uma outra tabela, para endereços ainda desconhecidos, como o de FIM no programa da Fig. 9.6(b). No final da montagem, quando estes endereços já são conhecidos, o montador vai completando esta tabela de endereços desconhecidos com os dados da tabela de símbolos. É uma tarefa adicional à montagem, mas não se trata de voltar ao início do programa. Consiste apenas em completar dados de uma tabela cujo valor já está conhecido.

O montador de um passo apresenta os seguintes problemas:

a) não conclui a tarefa com um código direto e seqüencialmente gerado. É preciso que haja um rearranjo dos endereços para se inserir os que estavam desconhecidos;

b) se a tabela de endereços desconhecidos foi grande (muitas referências a endereços ainda inexistentes), a busca às suas diversas entradas será demorada, talvez tão demorada quanto se tivesse realizado um segundo passo, com melhores resultados finais.

9.3.2 Compilação

Quando se pretende converter para linguagem de máquina um programa escrito em linguagem de mais alto nível que o da linguagem de montagem, então o método utilizado se chama *Compilação*.

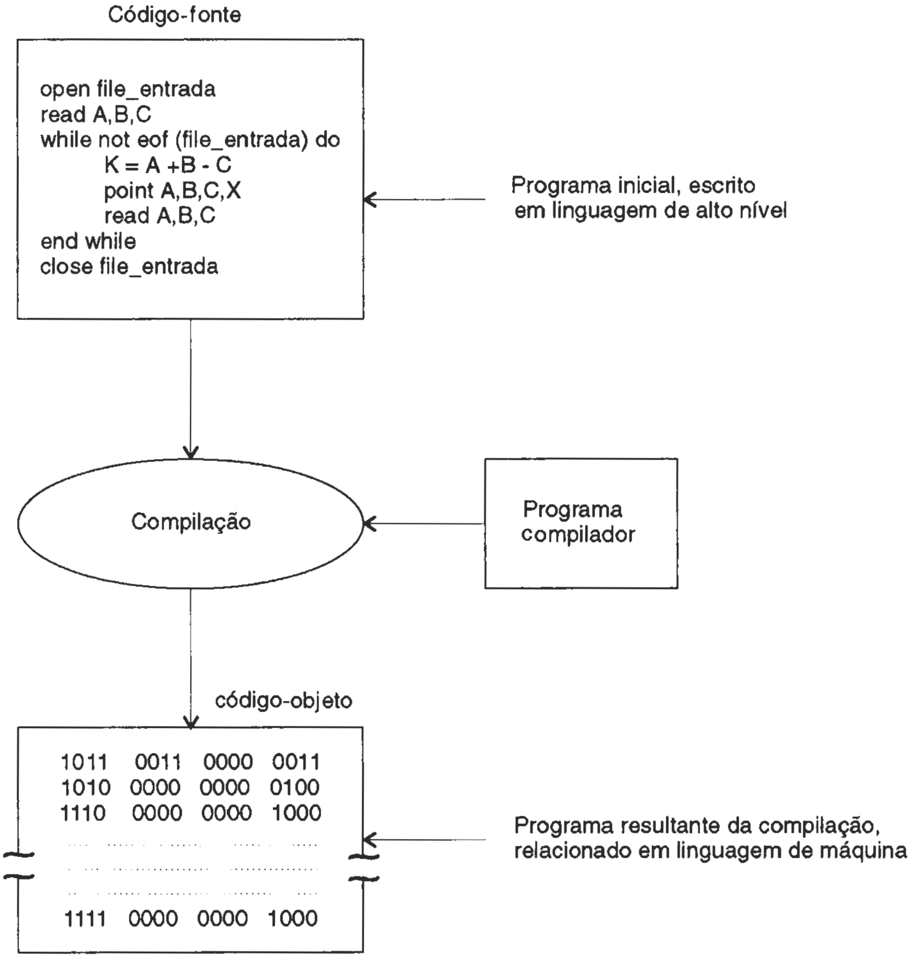
Compilação é, pois, o processo de análise de um programa escrito em linguagem de alto nível, o programa-fonte (ou código-fonte) e sua conversão (ou tradução, como alguns preferem) em um programa equivalente, porém descrito em linguagem binária de máquina, denominado programa-objeto (ou código-objeto). O programa que realiza esta tarefa é denominado compilador, e o fluxograma básico do processo de compilação está mostrado na Fig. 9.7.

A compilação é um processo semelhante ao de montagem (na análise de programa-fonte, criação de tabelas auxiliares e geração de código-objeto final em linguagem de máquina), porém mais complexo e demorado. Na montagem, há uma relação de 1:1 entre as instruções de linguagem de montagem e as instruções de máquina, enquanto na compilação isto não acontece, pois um único comando em Pascal, por exemplo, pode gerar várias instruções de máquina.

Durante a compilação, o código-fonte é analisado, comando por comando; o programa compilador realiza várias tarefas, dependendo do tipo de comando que ele esteja analisando. Se, por exemplo, ele estiver analisando um comando que esteja declarando uma variável, ele criará a respectiva entrada em sua tabela de símbolos.

Inicialmente, o compilador realizará uma análise do código-fonte (normalmente esta tarefa é realizada por um módulo de compilador denominado *front-end*), dividido em três partes funcionalmente distintas:

- análise léxica;
- análise sintática;
- análise semântica.


Código-fonte
Programa inicial, escrito em linguagem de alto nível
Compilação
Programa compilador
código-objeto
Programa resultante da compilação, relacionado em linguagem de máquina

```
open file_entrada
read A,B,C
while not eof (file_entrada) do
        K = A +B - C
        point A,B,C,X
        read A,B,C
end while
close file_entrada
```

```
1011  0011  0000  0011
1010  0000  0000  0100
1110  0000  0000  1000

1111  0000  0000  1000
```

Figura 9.7 Fluxograma básico do processo de compilação.

Após esta tripla análise, é gerado um código intermediário e são construídas várias tabelas, como a tabela de símbolos, que auxiliará a realização da segunda fase, que é a fase de efetiva criação do código binário de máquina, o *código-objeto*. Nesta segunda fase, o módulo responsável do compilador (usualmente conhecido como *back-end*) aloca memória para as variáveis e realiza a atribuição dos registradores a serem utilizados, além da geração do código-objeto final.

A função de *análise léxica* do compilador consiste em decompor o programa-fonte em seus elementos individuais distintos (comandos, operadores, variáveis, etc.), os quais são verificados de acordo com as regras da linguagem (por exemplo, em Pascal o sinal de igualdade é composto por dois símbolos, : =, e não apenas pelo símbolo =), gerando mensagem de erro se for encontrada alguma incorreção. Cada operador ou palavra-chave (comando) é identificado com um número pelo analisador léxico.

A função do *analisador sintático* de um compilador consiste basicamente na criação das estruturas de cada comando, na verificação da correção dessas estruturas e na alimentação da tabela de símbolos com as informações geradas. Ele realiza esta tarefa a partir de cada elemento obtido da análise léxica, montando a estrutura apropriada (em geral, é uma árvore) de acordo com as regras gramaticais da linguagem. A Fig. 9.8 mostra uma árvore de análise (*parse tree*) para o comando

X:= (a + b) * c + d;

a) Comando: X:= (a + b) * c + d ;

b) Regras gramaticais	Abreviatura
Expressão → Expressão + Termo	E → E + T
Expressão → Termo	E → T
Término → Fator * Termo	T → F * T
Fator → (Expressão)	F → (E)
Término → Fator	T → F
Fator → Identificador	F → U

c) Árvore de análise

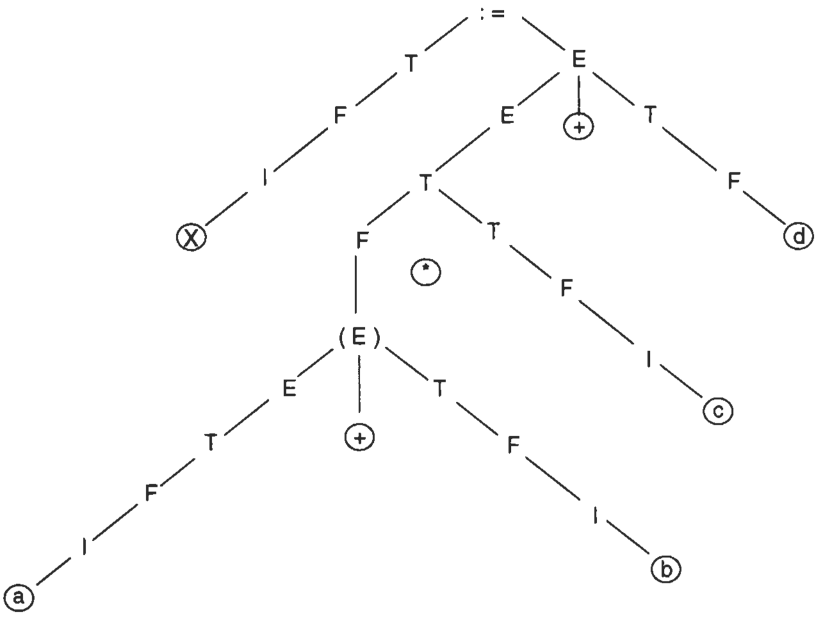


Figura 9.8 Árvore da análise do comando: X:= (a + b)* c + d;.

Na figura são apresentadas as regras gramaticais da linguagem, referentes aos elementos do comando exemplificado, bem como a árvore de análise resultante. Na realidade, a árvore mostrada é apenas uma parte da árvore completa, que representa o programa-fonte por inteiro.

Além de permitir, com esta estrutura, a futura criação de código de máquina, o analisador verifica a correção do comando em termos de sua conformidade com a regra gramatical especificada na definição da linguagem, gerando uma mensagem de erro quando identifica uma incorreção.

A tabela de símbolos gerada durante o processo contém entradas para cada identificador e cada literal usado no programa-fonte, com as informações de seus atributos. Por exemplo, os identificadores de dados (nome da variável usada) terão como atributo seu tipo, isto é, se é inteiro, se é fracionário, etc., de modo que a operação aritmética a ser realizada com ele siga o algoritmo apropriado (ver Cap. 7).

A análise semântica realizada pelo compilador verifica as regras semânticas estáticas da linguagem, produzindo, da mesma forma que os demais analisadores, mensagem de erro para as incorreções ou inconsistências semânticas.

Usa-se o termo *regra semântica estática* porque se refere a regras que podem ser verificadas durante o processo de compilação, diferençando-se de regras de semântica que só podem ser verificadas durante a execução do código de máquina.

Regras semânticas estáticas são, por exemplo, a necessidade de integridade de um tipo de dado em uma expressão, isto é, o tipo de dado mostrado na expressão tem que ser coerente com o que foi declarado. Uma declaração de GOTO não poder transferir controle para fora do módulo ou subprograma é um outro exemplo de regra semântica estática, como também usar o comando CASE, que deve conter todos os seus elementos.

Como já mencionado, o módulo de frente de um compilador (*front-end*) cria a tabela de símbolos durante a tripla análise do programa-fonte (léxica, sintática e semântica), inserindo nela as informações necessárias para o trabalho do módulo *back-end*, de alocação de memória, etc.

Por exemplo, considerando uma parte da *data division* de um programa COBOL, como a seguir ilustrado:

```
77 A PIC 9(6)
77 B PIC 9(6)
77 C PIC 9(6)
```

observamos que se trata da declaração de três variáveis do tipo inteiro, ocupando, cada uma, 6 algarismos decimais (ver representação BCD no item 7.2.6). Ao encontrá-las em sua análise, o compilador cria três entradas na tabela de símbolos, conforme mostrado na Fig. 9.9, cada uma contendo o nome da variável, seu tipo, tamanho e endereço de memória. O tamanho, em bytes, é obtido conforme o tipo de dado e sua declaração (no exemplo da Fig. 9.9, o tamanho será de 4 bytes porque se trata de tipo BCD, decimal, que representa cada algarismo com 4 bits (um valor com 6 algarismos gastará 6 × 4 = 24 bits ou 3 bytes, mais 1 byte para representar o sinal do número)). O endereço de memória é calculado em função do espaço ocupado pelo dado (no exemplo, seriam 4 bytes ou 4 células de memória, por isso cada endereço é defasado do anterior de 4 unidades).

O módulo *back-end* de um compilador completa o processo, alocando espaço de memória, definindo que registradores serão utilizados e que dados serão armazenados neles, e gerando o código-objeto final, em linguagem binária de máquina.

Nome	Tipo	Tamanho	Endereço
A	Interno	4	1000
B	Interno	4	1004
C	Interno	4	1008

Figura 9.9 Exemplo de tabela de símbolos.

O gerador de código é a parte do módulo que converte a representação intermediária, gerada pelo módulo *front-end*, em código-objeto. O código-objeto pode ser *absoluto*, isto é, os endereços constantes do código são endereços reais de memória, ou pode ser *relocável*, isto é, os endereços são relativos ao início do programa, transformando-se em endereços reais apenas na ocasião de execução. Códigos relocáveis são mais versáteis e, por isso, mais utilizados. Eles podem ser compilados separadamente e, posteriormente, ou logo em seguida, ligados (linkeditados) para formar um módulo único que será executado.

Como, em geral, o conjunto de instruções das UCP modernas possui uma grande quantidade de instruções, algumas bem semelhantes (há, por exemplo, mais de 10 instruções de soma nos processadores da família Intel x86 e 27 nos processadores VAX-11), o gerador de código pode produzir diferentes seqüências de código de máquina para um mesmo cálculo. Na prática, os compiladores ignoram muitas das instruções existentes, sendo este pouco uso uma das desvantagens alegadas para máquinas deste tipo (CISC) em relação aos processadores com arquitetura de poucas instruções (RISC) (ver Cap. 11).

9.4 LIGAÇÃO OU LINKEDIÇÃO

Quando um programador escreve um programa (qualquer que seja a linguagem utilizada), ele não se preocupa em codificar determinadas operações, porque o código binário necessário para realizar aquelas tarefas já existe armazenado no sistema. É preciso apenas que o código em questão seja localizado e buscado onde estiver e incorporado ao programa. É o caso, por exemplo, de comandos de entrada e saída. Qualquer que seja o programa a ser desenvolvido, se nele foi necessário realizar uma impressão de informações, as ações a executar serão sempre as mesmas, pois o hardware é o mesmo (a impressora), qualquer que seja o programa. É claro que alguns elementos são diferentes em cada programa (por exemplo, no caso de impressão, cada programa imprimirá uma quantidade de bytes diferentes, podendo ser arquivos diferentes, armazenados em locais diferentes da memória, etc. Portanto, estas informações têm que ser passadas ao código comum de controle de impressão).

Podem ser citados outros exemplos referentes a determinadas rotinas que não fazem parte do programa de aplicação, mas devem ser incorporados a ele para efetiva execução. Um desses exemplos pode ser o de certas rotinas matemáticas especiais, como o cálculo de um *seno* ou outra operação trigonométrica qualquer.

As rotinas externas ao programa (assim chamadas porque não fazem parte dele) são normalmente organizadas em arquivos (cada rotina consta de um arquivo de código binário), que constituem diretórios específicos para cada grupo de rotinas (diretório com rotinas matemáticas, diretório com rotinas de entrada/saída, e outros). Esses diretórios são usualmente denominados *bibliotecas* (*libraries*). Uma biblioteca é uma coleção de códigos-objeto, um para cada rotina específica, e é indexada pelo nome da rotina. Quando o programa de aplicação deseja usar uma dessas rotinas, ele insere uma chamada de biblioteca no seu código (*library call*).

Do que foi descrito pode-se concluir que um código-objeto gerado por um compilador não é imediatamente executável, visto que ainda há código binário a ser incluído no programa, como uma chamada de rotina de impressão, ou uma chamada de rotina para cálculo de um seno, etc.

Quando o compilador, ao gerar o código-objeto, encontra um comando da linguagem de alto nível que exige o uso de uma rotina de biblioteca, ele insere uma chamada para a rotina em código-objeto. Esta chamada (CALL) inclui o nome da rotina, bem como o endereço de qualquer dado que deve ser passado entre o programa e a rotina.

Por exemplo, considere o seguinte comando Pascal:

```
X: = A * B;
```

Suponha que A, B e X sejam números representados em ponto flutuante e que o compilador somente suporte aritmética de ponto flutuante via rotina externa, armazenada em uma biblioteca do compilador. E que, por esta razão, o compilador irá inserir no código-objeto a seguinte chamada:

```
CALL MPY_FP (1AB5, 1AB9, 1ABF)
```

Este nome, MPY_FP, não é uma instrução de máquina, mas sim um índice para a biblioteca, onde se encontra um programa, já compilado, que realiza a operação aritmética de multiplicação segundo o algoritmo de ponto flutuante. Os números entre parênteses são os endereços de memória dos dados A e B e do resultado X.

Para que o programa seja executado, é necessário que a rotina MPY_FP seja incorporada ao código-objeto do programa de aplicação, isto é, que haja uma conexão lógica entre o código-objeto principal e o código-objeto de rotina. Este processo de interpretação da chamada (CALL) e a respectiva conexão com a rotina chamada denomina-se *ligação* ou *linkedição* e é realizado por um programa *ligador* ou *linkeditor.*

A Fig. 9.10 mostra o fluxograma básico do processo de execução de um programa que inclui as duas etapas já descritas: a *compilação* e a *ligação.* Como resultado do processo de ligação, obtém-se um conjunto de códigos de máquina, interligados e prontos para execução. Chama-se este conjunto de códigos de *módulo de carga* ou *código executável* (p. ex., os arquivos gerados no sistema operacional DOS que possuem a terminação EXE e COM são códigos executáveis, enquanto os arquivos com terminação OBJ são códigos-objeto).

Na terminologia comumente adotada para o assunto de ligação, a chamada CALL MPY_ FP é denominada *referência externa não-resolvida.* Como se trata de uma chamada para uma rotina que não pertence ao código-objeto que está sendo gerado pelo compilador, sua localização é desconhecida para ele, por isso diz-se "não-resolvida", e externa porque não pertence ao código-objeto referido.

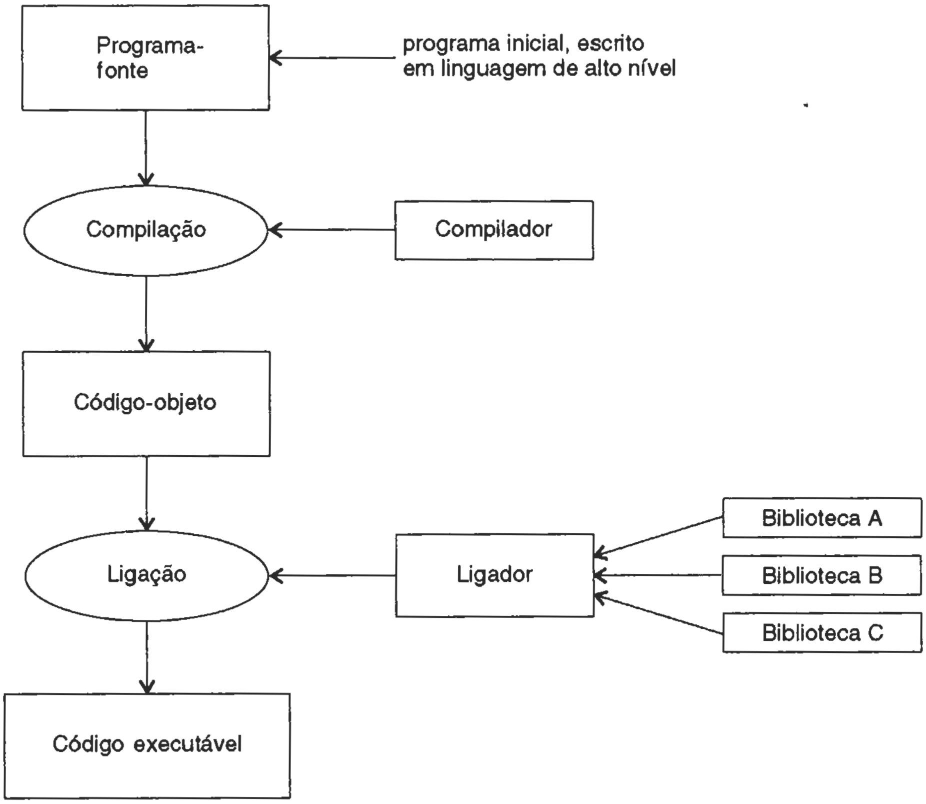


Figura 9.10 Fluxograma do processo de compilação e ligação.

O funcionamento do programa ligador consiste em examinar todo o código-objeto, gerado após a compilação, e procurar as referências externas não-resolvidas. Para cada uma, procurar sua localização nas bibliotecas indicadas nos comandos de execução do programa (estes comandos são diretivas para o sistema operacional realizar seu serviço, como, por exemplo, chamar o compilador e iniciar esta etapa, chamar o ligador, etc.).

Quando o ligador encontra a rotina chamada em uma das bibliotecas, ele substitui a linha de chamada pelo código-objeto da rotina. Isto se chama "resolver a referência externa". Uma das possíveis mensagens de erro que podem surgir após a conclusão do processo de ligação é a de não se ter encontrado alguma rotina. Nesse caso, o código-objeto continua sem poder ser executado, pois ainda haverá uma (ou mais de uma) referência externa não-resolvida.

Na prática, tanto o processo de compilação quanto o de ligação são fases distintas e independentes do procedimento global de execução de um programa, embora possam ser realizadas em seqüência e imediatamente uma após a outra, como se fossem uma única atividade. Por serem independentes, também geram produtos distintos: o código-objeto (ao final da compilação) e o código executável (após a ligação), os quais são arquivos de códigos binários que podem ser armazenados em memória secundária para uso imediato ou posterior.

Em outras palavras, a execução de um programa se inicia pelo armazenamento, na memória principal, do código-fonte e do programa compilador e este, após a realização de sua tarefa, gera o código-objeto, criando um arquivo com o nome de programa-objeto (o sistema operacional DOS de microcomputadores costuma usar a terminação OBJ para completar os nomes dos arquivos que contêm código-objeto). Este arquivo pode ser armazenado na memória secundária para ser ligado mais tarde ou ser imediatamente carregado na memória principal, juntamente com o programa ligador, e se iniciar a etapa de ligação. A decisão do que fazer depende das diretivas definidas pelo programador quando executa o programa. Da mesma forma que a compilação, após a fase de ligação, está formado o código executável ou módulo de carga. Este código é também armazenado em um arquivo, que pode ser imediatamente carregado na memória para ser executado efetivamente pela UCP ou pode ser armazenado na memória secundária para uso posterior.

Por exemplo, uma empresa desenvolve um sistema que vai calcular a sua folha de pagamento mensal (na realidade, é um conjunto de programas). O processo de desenvolvimento inclui a definição dos algoritmos e a codificação dos programas em linguagem de alto nível. Após a codificação, cada programa é compilado algumas vezes, pois, em geral, são descobertos erros de codificação. Tão logo todos os programas estejam com seus códigos-objeto corretos (não se encontram mais erros — os famosos *bugs*), o sistema é integrado pelo processo de ligação, que produz o código executável final (também obtido após correção de eventuais erros).

O código executável é, então, executado algumas vezes com uma massa de dados de teste para verificação da correção da lógica do sistema, corrigindo-se eventuais erros surgidos nessa ocasião, e, finalmente, ele é armazenado na memória secundária, sendo recuperado para execução sempre que a folha de pagamento da empresa tiver que ser calculada. Desta forma, não se perde mais tempo com compilação e ligação.

Há também um tipo de ligador que não cria o código executável, sendo, portanto, um pouco mais rápido. Chama-se carregador, ou *loader*. O programa carregador realiza em seqüência imediata as duas tarefas: ligação e execução do código de máquina, sem geração de código executável permanente. Ele apenas cria o executável, sem armazená-lo, e imediatamente inicia a execução.

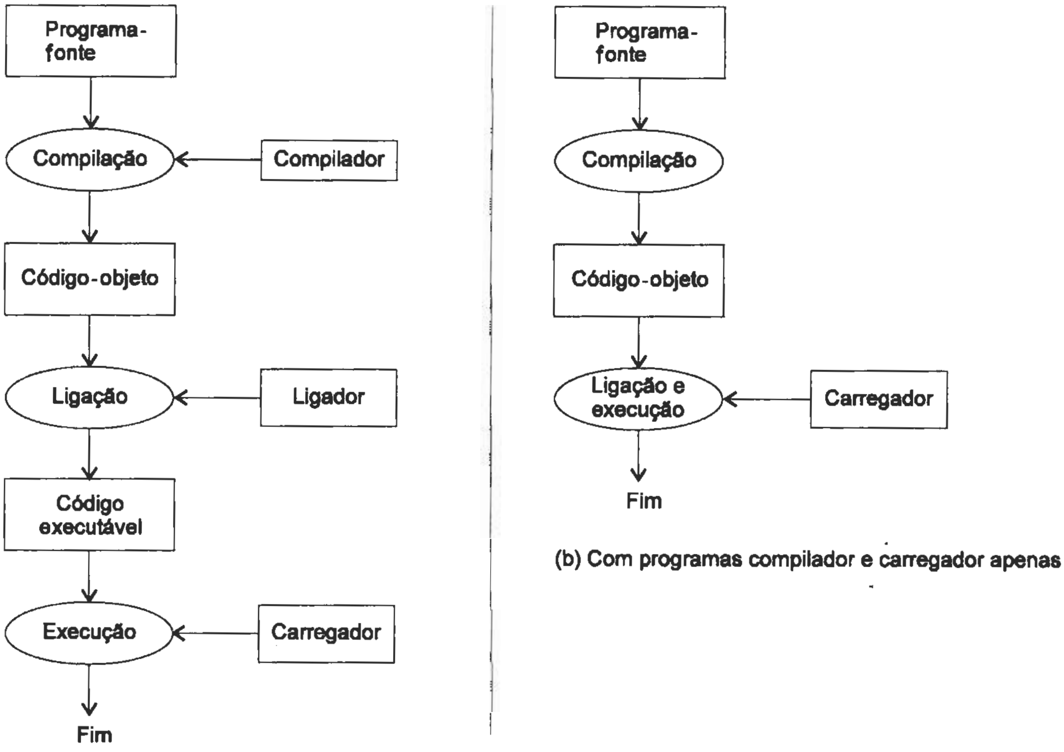
9.5 INTERPRETAÇÃO

O processo de execução de um programa através das três fases distintas, descritas nos itens anteriores, é apresentado no fluxograma da Fig. 9.11. No entanto, este não é o único método de execução de um programa. Há um outro processo, denominado *interpretação*, que, embora com o mesmo resultado final, apresenta o modo de realização da interpretação bastante diverso do método compilação /ligação/ execução.

Com o método compilação/ ligação /execução, para que um programa possa ser efetivamente executado é necessário que todos os comandos do código-fonte desse programa sejam previamente convertidos para código-objeto e este tenha tido todas as referências externas resolvidas (etapa de ligação). A compilação não compreende execução; ela é apenas uma fase de tradução, de conversão. Além disso, o método gera produtos bem distintos, como o *código-objeto* e, mais tarde, o código executável.

Em contrapartida, o método de interpretação se caracteriza por realizar as três fases (compilação, ligação e execução), comando a comando, do programa-fonte. Não há, pois, um processo explícito de compilação e ligação. Na realidade, um programa-fonte é diretamente executado (interpretado) por um outro programa (o interpretador) e produz o resultado. Não há produtos intermediários, como o código-objeto ou código executável, como acontece no método anterior.

Em resumo, pelo método de interpretação, cada comando do código-fonte é lido pelo interpretador, é convertido em código executável e imediatamente executado, antes que o comando seguinte seja lido.



(b) Com programas compilador e carregador apenas

(a) Com programas compilador, ligador e carregador distintos

Figura 9.11 Fluxograma do processo de execução completa de um programa.

Há linguagens de programação cujas características estruturais são típicas de métodos de compilação, possuindo, portanto, apenas compiladores. São exemplos deste tipo: Cobol, Fortran, C e Pascal. Há outras que possuem apenas interpretadores, como o Apl. A linguagem Basic foi durante algum tempo utilizada apenas com interpretadores, porém já há algum tempo foram desenvolvidos compiladores para o Basic, de modo que o usuário atualmente pode optar por um ou outro tipo. A linguagem Java é interpretativa.

9.5.1 Compilação × Interpretação

Ambos os métodos possuem vantagens e desvantagens, oriundas do modo próprio de funcionamento de cada um. A principal vantagem da interpretação sobre a compilação é sua capacidade de identificar e indicar um erro no programa-fonte, seja na etapa de conversão da fonte para executável (estática), seja na execução do código binário (dinâmica), isto é, erro na lógica do algoritmo ou na inconsistência entre o valor do dado e o tipo de dado definido, por exemplo. Uma razoável desvantagem da interpretação é o consumo de memória.

No que se refere ao consumo de memória, verificamos que o método de compilação usa memória apenas por períodos definidos de tempo. Ou seja, o compilador só permanece na memória durante a fase de compilação; *ao terminar esta fase, o compilador cede espaço para o ligador e este, em seguida, cede espaço para o* carregador executar o código binário.

Em compensação, o programa interpretador deve permanecer na memória durante toda a execução do programa, porque cada comando necessita do interpretador. E estes são programas grandes, que ocupam uma área considerável de memória.

Uma outra desvantagem da interpretação sobre a compilação consiste na possibilidade de certas partes do código de um programa-fonte (um loop, por exemplo) terem que ser interpretadas tantas vezes quantas definidas no loop, enquanto, no método de compilação, isto sempre acontece uma única vez.

Por exemplo, consideremos o trecho de programa a seguir, que contém um loop:

```
FOR  J = 1 TO 1000
   BEGIN
        READ A,B;
        X: = A+B;
        PRINT X;
   END
END
```

No método de compilação, os três comandos dentro do loop (juntamente com os demais comandos) serão convertidos uma única vez para código executável (objeto e depois executável) e, embora sejam realizadas 1000 leituras de A e B, 1000 somas e 1000 impressões, só ocorre uma tradução de código.

Pelo método de interpretação, haveria 1000 conversões de fonte para executável, 999 a mais do que na técnica anterior, o que consumiria mais UCP, aumentando o tempo total de execução do programa em relação à mesma execução pelo método de compilação. Apesar de já existirem interpretadores mais eficientes, que não realizam todas as 999 interpretações, ainda assim, nesse particular, não são tão eficazes quanto os compiladores.

O mesmo problema ocorre com programas que são executados freqüentemente. Vejamos, por exemplo, o sistema de folha de pagamento a que nos referimos anteriormente. No caso de se usar o método de compilação, haverá uma única tradução (compilação e ligação) e periodicamente só será executado o módulo de carga (código executável). Com a técnica de interpretação, o programa seria convertido toda vez que precisasse ser executado, o que gastaria sempre mais tempo.

A Tabela 9.2 mostra um quadro comparativo que resume as considerações relativas ao consumo de recursos de computação com os processos de compilação e interpretação feitas anteriormente.

Tabela 9.2 Resumo do Uso de Recursos de Computação durante o Processo de Compilação e de Interpretação

Recursos	Compilação	Interpretação
Uso da memória (durante a execução)		
— Interpretador ou compilador	Não	Sim
— Código-fonte	Não	Parcial
— Código executável	Sim	Parcial
— Rotinas de bibliotecas	Só as necessárias	Todas
Instruções de máquina (durante a execução)		
— Operações de tradução	Não	Sim
— Ligação de bibliotecas	Não	Sim
— Programa de aplicação	Sim	Sim

Os interpretadores são, no entanto, bastante vantajosos quando se trata de desenvolvimento de programas e correção de erros nesta fase.

A Fig. 9.12 mostra os fluxogramas de etapas utilizadas durante o desenvolvimento e depuração de erros em programas-fonte para ambos os métodos: compilação e interpretação.

Quando se utiliza o método de compilação, a identificação de erros no programa se torna mais problemática à medida que o código executável entra em fase de execução. Ou seja, parece difícil identificar exatamente a origem do erro, pois não há uma relação entre o comando do código-fonte e as instruções de máquina do código executável. No fonte, temos, por exemplo, nomes simbólicos de variáveis, enquanto no executável há endereços de memória (onde as variáveis estão armazenadas) e um nome de comando (Do while, por exemplo) que é substituído por um ou mais códigos de operação numéricos. Erros de execução são, às vezes, bem difíceis de identificar devido justamente à falta de uma relação mais bem definida entre fonte e executável. Se,

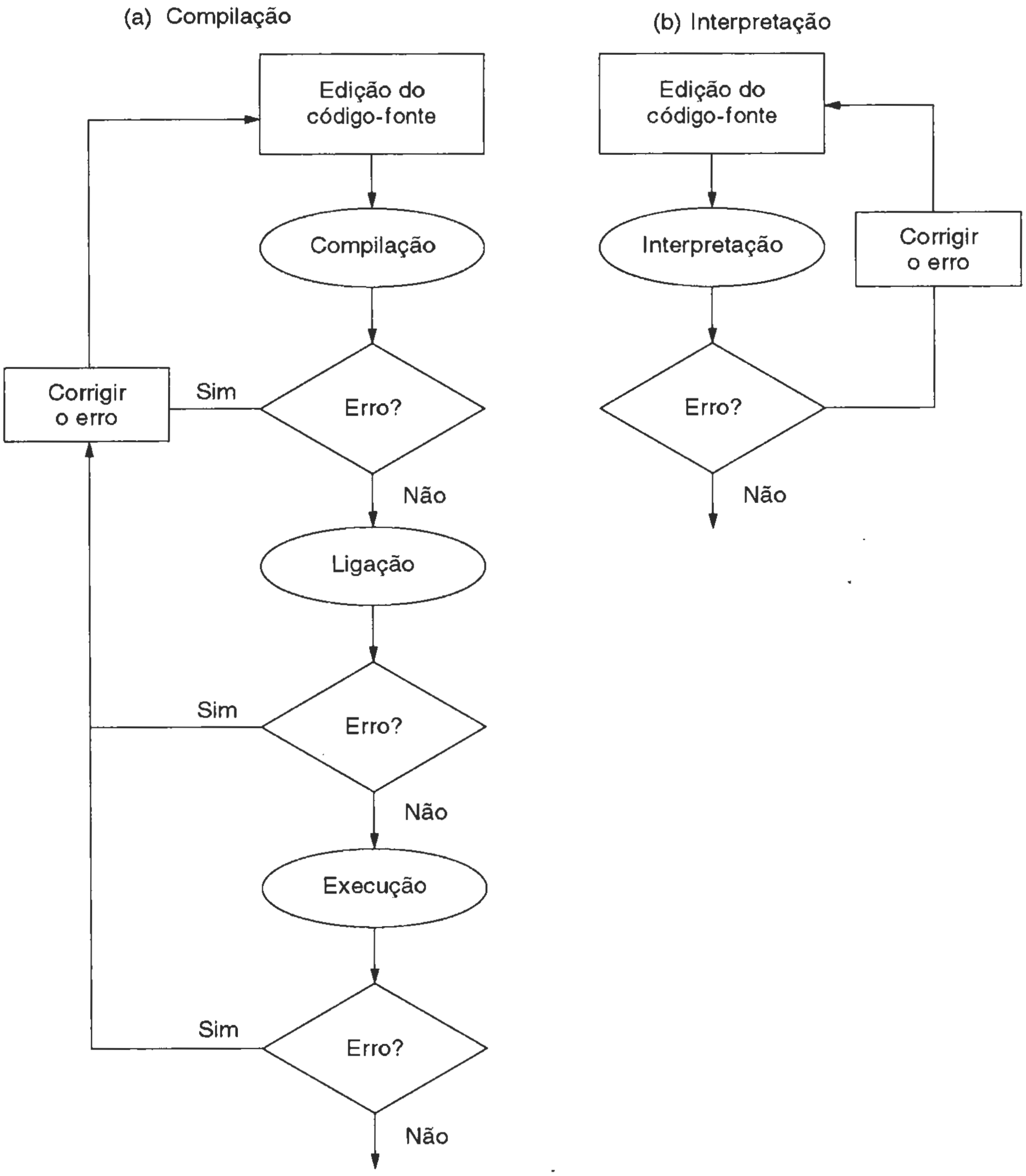

(a) Compilação
Edição do código-fonte
Compilação
Erro?
Sim
Corrigir o erro
Não
Ligação
Erro?
Sim
Não
Execução
Erro?
Sim
Não
(b) Interpretação
Edição do código-fonte
Interpretação
Corrigir o erro
Erro?
Não

Figura 9.12 Fluxograma do processo de desenvolvimento e depuração de programas, utilizando os métodos de compilação (a) e interpretação (b).

por exemplo, ocorrer um erro do tipo divisão por zero, a mensagem de erro conterá como informação o endereço de memória da instrução que resultou no erro. O programador terá de identificar que comando do código-fonte gerou o referido erro.

Por outro lado, quando se emprega o método de interpretação, é mais simples a relação entre o código-fonte e o executável, porque cada comando-fonte é imediatamente traduzido e executado. Assim, se ocorrer um erro, o responsável deve ser o comando que está sendo executado e, portanto, já está identificado. O interpretador pode informar o erro, indicando diretamente o comando, ou variável, causador da ocorrência, pelo seu nome simbólico e não por um endereço numérico de memória.

9.6 EXECUÇÃO DE PROGRAMAS EM CÓDIGO DE MÁQUINA

Antes de iniciar este item, vamos observar, por meio de um exemplo, as diferenças relativas ao desenvolvimento de programas em linguagem de alto nível e em linguagem Assembly ou de máquina. Essas diferenças

```
Início do programa principal
X recebe 1
Y recebe 2
Z recebe a soma de X com Y
Se Z maior que zero
                    Então: Z recebe o Quadrado (X)
                    Senão: Z recebe o Quadrado (Y)
Fim do Se
Fim do programa principal

Função do Quadrado:
                    Início função do Quadrado
                    Retorne o produto do parâmetro da função com ele mesmo
Fim da função do Quadrado
```

Figura 9.13(a) Pseudocódigo de um algoritmo.

podem ser verificadas no que se refere ao tamanho dos programas em cada linguagem e ao diferente grau de dificuldade que o leitor poderá encontrar na simples observação de cada programa. Para tanto, foi criado um algoritmo bem simples, que trata apenas da soma de dois valores (uma operação aritmética), realiza uma operação de desvio condicional e se utiliza de uma função, conforme mostrado na Fig. 9.13(a). Em seguida, o referido algoritmo foi codificado em 2 linguagens de alto nível, em Pascal, mostrado na Fig. 9.13(b), e em C, mostrado na Fig. 9.13(c).

O referido algoritmo também foi codificado na linguagem de montagem do Intel 8088/486, utilizando-se o montador do DOS 5.0. Finalmente, os três programas foram compilados e montados (os dois primeiros compilados e o último montado), e seus resultados em linguagem de montagem gerada pelos compiladores e montador constam das Figs. 9.13(a)(b), 9.13(a)(c) e 9.13(a)(d), respectivamente.

No Cap. 6 foram descritos, de forma detalhada, os passos necessários para se completar a realização do ciclo de instrução de algumas instruções de máquina. Neste capítulo, foram apresentados processos de conversão de um programa criado em uma linguagem de alto nível (do tipo Pascal, C, Cobol, Delphi, etc.) para código binário executável diretamente pela máquina, como também de programas escritos em linguagem de montagem.

Para completar o assunto, vamos apresentar a execução propriamente dita de um programa, com a intenção de mostrar a sistemática e metódica seqüência de execução de ciclos de instrução, que caracteriza a ligação UCP/MP.

```
                              EXAMPLE1.PAS
(*
* EXAMPLE1.PAS                Exemplo em linguagem Pascal
*)

Program Example1;
Var
x, y, z       : integer;
function func (w: integer) : integer;
Begin
          Func: = w * w;
End;
Begin
          x: = 1;
          y: = 2;
          z: = x + y;
          if (z > 0)
             then z: = func (x)
          else z: = func (y);
End
```

Figura 9.13(b) Programa em Turbo Pascal Versão 5.0, que implementa o algoritmo da Fig. 9.13(a).

```
int func (int w);

void main (void)
{
                    int x, y, z;

x = 1;
y = 2;
z = x + y;
if (z > 0)
                    z = func (x);
else z = func (y);
}
int func (int w);
{
return (w * w);
}
```

Figura 9.13(c) Implementação do algoritmo da Fig. 9.13(a) em Turbo C++.

Por fim, vamos descrever a efetiva execução, pela UCP, de um pequeno programa. Para simplificar o processo, vamos utilizar o processador cujas características foram apresentadas na Fig. 6.19 e acompanhar, passo a passo, a realização dos sucessivos ciclos de instrução concernentes à completa execução do programa.

Consideremos a expressão matemática

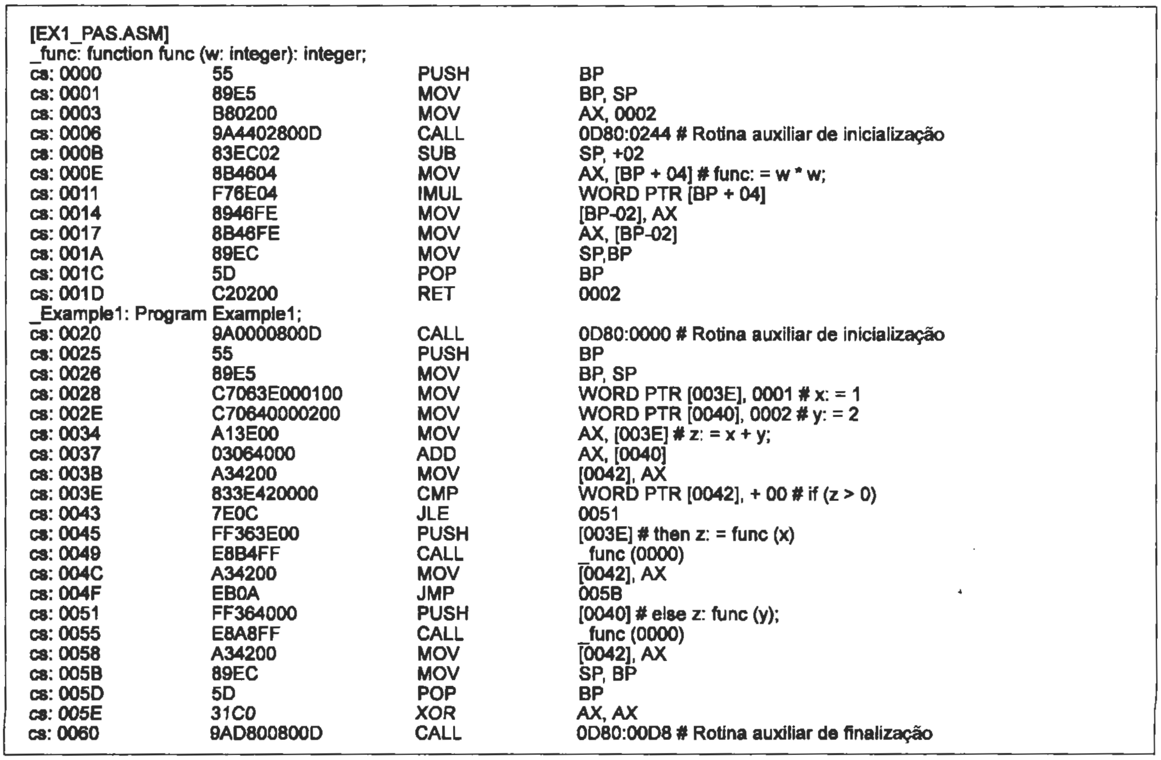
$$X = Y + Z - T$$

que imprime o valor de X, se este não for igual a zero.

```
                    EXAMPLE1.ASM
# EXAMPLE1.ASM                              Exemplo em linguagem Assembly
                    push ax                 # salva os registradores a serem usados
                    push bx                 # na rotina
                    push cx
                    push dx
                    mov cx, 0001            # x = 1
                    mov dx, 0002            # y = 2
                    mov ax, cx              # z = x + y
                    add ax, dx
                    mov bx, ax
                    or bx, bx               # if (z > 0)
                    jle JUMP 1
                    push cx                 # then x como parâmetro
                    jmp JUMP 2

JUMP1:              push dx                 # else y como parâmetro
JUMP2:              call CALL 1             # Chama rotina int func (parâmetro)
                    pop bx                  # z = func (parâmetro)
                    pop dx                  # Restaura os registradores usados
                    pop cx
                    pop bx
                    pop ax
                    ret                     # Fim da rotina
CALL1:              push bp                 # Início da rotina int func (parâmetro)
                    mov bp, sp
                    mov ax, [bp + 04]       # Recupera parâmetro
                    imul ax                 # Calcula produto
                    mov [bp + 04], ax       # Prepara valor de retorno
                    pop bp
                    ret                     # Fim da rotina int func (parâmetro)
```

Figura 9.13(d) Programa em linguagem Assembly, que implementa o algoritmo mostrado na Fig. 9.13(a).



```
[EX1_PAS.ASM]
_func: function func (w: integer): integer;
cs: 0000          55              PUSH     BP
cs: 0001          89E5            MOV      BP, SP
cs: 0003          B80200          MOV      AX, 0002
cs: 0006          9A4402800D      CALL     0D80:0244 # Rotina auxiliar de inicialização
cs: 000B          83EC02          SUB      SP, +02
cs: 000E          8B4604          MOV      AX, [BP + 04] # func: = w * w;
cs: 0011          F76E04          IMUL     WORD PTR [BP + 04]
cs: 0014          8946FE          MOV      [BP-02], AX
cs: 0017          8B46FE          MOV      AX, [BP-02]
cs: 001A          89EC            MOV      SP,BP
cs: 001C          5D              POP      BP
cs: 001D          C20200          RET      0002
_Example1: Program Example1;
cs: 0020          9A0000800D      CALL     0D80:0000 # Rotina auxiliar de inicialização
cs: 0025          55              PUSH     BP
cs: 0026          89E5            MOV      BP, SP
cs: 0028          C7063E000100    MOV      WORD PTR [003E], 0001 # x: = 1
cs: 002E          C70640000200    MOV      WORD PTR [0040], 0002 # y: = 2
cs: 0034          A13E00          MOV      AX, [003E] # z: = x + y;
cs: 0037          03064000        ADD      AX, [0040]
cs: 003B          A34200          MOV      [0042], AX
cs: 003E          833E420000      CMP      WORD PTR [0042], + 00 # if (z > 0)
cs: 0043          7E0C            JLE      0051
cs: 0045          FF363E00        PUSH     [003E] # then z: = func (x)
cs: 0049          E8B4FF          CALL     _func (0000)
cs: 004C          A34200          MOV      [0042], AX
cs: 004F          EB0A            JMP      005B
cs: 0051          FF364000        PUSH     [0040] # else z: func (y);
cs: 0055          E8A8FF          CALL     _func (0000)
cs: 0058          A34200          MOV      [0042], AX
cs: 005B          89EC            MOV      SP, BP
cs: 005D          5D              POP      BP
cs: 005E          31C0            XOR      AX, AX
cs: 0060          9AD800800D      CALL     0D80:00D8 # Rotina auxiliar de finalização
```

Figura 9.13(ab) Programa gerado pelo compilador Pascal relativo à compilação do programa mostrado na Fig. 9.13(b).

Poderia ser criado um programa em uma linguagem de alto nível do tipo Fortran ou C que resolvesse a referida expressão e produzisse a impressão de X sempre que a condição estabelecida fosse verdade. O programa completo poderia ser semelhante ao mostrado na Fig. 9.14.

Para que seja possível a execução desse programa, há necessidade de se utilizar uma instrução de desvio condicional, já que aparece uma condição a ser satisfeita. Ou seja, somente será impresso o valor de X se este valor for diferente de zero (a condição especificada é: se $X \neq 0$); caso contrário (ELSE), esse programa termina SEM imprimir o valor de X.

Em primeiro lugar, vamos repetir a definição de *instrução de desvio*, apresentada no Cap. 6 (ver Fig. 6.19), e caracterizar a diferença entre *desvio incondicional* e *desvio condicional*, de modo que possamos melhor entender as instruções de desvio mostradas naquela figura.

Desvio é uma alteração forçada da seqüência de execução de um programa. Em outras palavras, sendo o hardware da UC projetado para, após a busca de uma instrução, incrementar o conteúdo do CI e apontar para a instrução imediatamente seguinte, o desvio *é* a possibilidade de alterarmos o conteúdo do CI, de modo a armazenar-se nesse registrador um outro valor de endereço (que não o da próxima instrução na seqüência). Em outras palavras, consiste em se poder alterar a seqüência de realização de ciclos de instrução.

Com *desvio incondicional*, não há condição a ser satisfeita e o desvio é sempre executado; o programa tem alterada a sua ordem normal de execução, desviando-se para uma instrução fora da seqüência, independentemente de qualquer outra circunstância.

O resultado da execução de uma instrução de desvio incondicional é que, no final da sua execução, o CI conterá o valor existente no campo operando da instrução (endereço do desvio).

Com *desvio condicional*, o valor existente no campo do operando da instrução somente será transferido para o CI (execução do desvio) *se* uma dada condição for satisfeita, como, por exemplo, se $ACC = 0$, se $ACC > 0$ ou se $ACC \neq 0$. Caso contrário, a seqüência de execução permanece inalterada (sem desvio, a UC comanda a busca da próxima instrução imediatamente seguinte).

```
                                                EX1_C.ASM
_main: void main (void)
cs: 02C2            55                          PUSH        BP
cs: 02C3            88RC                        MOV         BP, SP
cs: 02C5            83EC02                      SUB         SP, + 02
cs: 02C8            56                          PUSH        SI
cs: 02C9            57                          PUSH        DI
# EXAMPLE1#11:    x = 1;
cs: 02CA            BF0100                      MOV         DI, 0001
#EXAMPLE1#12:     y = 2;
cs: 02CD            C746FE0200                  MOV         WORD PTR [BP - 02], 0002
#EXAMPLE1#13:     z = x + y;
cs: 02D2            8BC7                        MOV         AX, DI
cs: 02D4            0346FE                      ADD         AX, [BP - 02]
cs: 02D7            8BF0                        MOV         SI, AX
#EXAMPLE1#14      If (z > 0)
cs: 02D9            0BF6                        OR          Si, SI
cs: 02DB            7E03                        JLE         #EXAMPLE#16 (02E0)
#EXAMPLE1#15:     z = func (x);
cs: 02DD            57                          PUSH        DI
cs: 02DE            EB03                        JMP         02E3
#EXAMPLE#16       z = func (y);
cs: 02E0            FF76FE                      PUSH        [BP - 02]
cs: 02E3            E80900                      CALL        _func (02EF)
cs: 02E6            59                          POP         CX
cs: 02E7            8BF0                        MOV         SI, X
#EXAMPLE#17: }
cs: 02E9            5F                          POP         DI
cs: 02EA            5E                          POP         SI
cs: 02EB            8BE5                        MOV         SP, BP
cs: 02ED            5D                          POP         BP
cs: 02EE            C3                          RET
_func: int func (int w)
cs: 02EF            55                          PUSH        BP
cs: 02F0            8BEC                        MOV         BP, SP
cs: 02F2            8B5E04                      MOV         BX, [BP + 04]
#EXAMPLE#21       return (w * w);
cs: 02F5            8BC3                        MOV         AX, BX
cs: 02F7            F7EB                        IMUL        BX
cs: 02F9            EB00                        JMP         02FB
#EXAMPLE1#22: }
cs: 02FB            5D                          POP         BP
cs: 02FC            C3                          RET
```

Figura 9.13(ac) Programa gerado pelo compilador C relativo à compilação do programa mostrado na Fig. 9.13(c).

A Fig. 6.19 mostra exemplos de instruções de desvio incondicional (JMP Op.) e de desvio condicional (JP Op., JN Op. e JZ Op.). Os processadores Intel 80486 e Pentium possuem várias instruções de desvio, como:

JA, JB, JG, JE, JZ, JNE, JS, JCXZ, JNL

Os processadores VAX-11 possuem cerca de 29 instruções de desvio (denominadas genericamente Branch e não Jump, como nos processadores Intel).

Vamos também utilizar no programa uma instrução de E/S, definida na Fig. 6.19. Trata-se da instrução PRT, que significa imprimir o valor armazenado do ACC.

A Fig. 9.15 mostra o programa em linguagem de montagem (Assembly) para resolver a equação dada (equivalente ao programa da Fig. 9.14).

Para detalhar a execução desse programa diretamente pelo computador, isto é, acompanhando cada ciclo de instrução, com os fluxos de controle, endereço e dados entre a UCP e a MP, precisamos primeiro converter o programa Assembly em outro, correspondente, porém totalmente em linguagem binária de máquina (na realidade, utilizaremos os valores em hexadecimal e não em binário, para simplificar os números e facilitar um pouco o entendimento).

Para converter o referido programa Assembly em linguagem de máquina e podermos realizar os ciclos de instrução passo a passo (execução do programa), vamos considerar que:

a) o processador/MP utilizado possui as mesmas características definidas na Fig. 6.19, inclusive as mesmas instruções (mesmos códigos de operação);

```
                              EX1_ASM.ASM

cs: 0100    50        PUSH   AX               #Salva os registradores a serem
cs: 0101    53        PUSH   BX               #usados na rotina
cs: 0102    51        PUSH   CX
cs: 0103    52        PUSH   DX
cs: 0104    B90100    MOV    CX, 0001         # x = 1
cs: 0107    BA0200    MOV    DX, 0002         # y = 2
cs: 010A    89C8      MOV    AX, CX           # z = x + y
cs: 010C    01D0      ADD    AX, DX
cs: 010E    89C3      MOV    BX, AX
cs: 0110    09DB      OR     BX, BX           #If (z >0)
cs: 0112    7E03      JLE    0117
cs: 0114    51        PUSH   CX               #Then x como parâmetro
cs: 0115    EB01      JMP    0118
cs: 0117    52        PUSH   DX               #Else y como parâmetro
cs: 0118    E80600    CALL   0121             #Chama rotina int func (parâmetro)
cs: 011B    5B        POP    BX               #z = func (parâmetro)
cs: 011C    5A        POP    DX               #Restaura os registradores usados
cs: 011D    59        POP    CX
cs: 011E    5B        POP    BX
cs: 011F    58        POP    AX
cs: 0120    C3        RET                     #Fim da rotina
cs: 0121    55        PUSH   BP               #Início da rotina int func (parâmetro).
cs: 0122    89E5      MOV    BP, SP
cs: 0124    8B4604    MOV    AX, [BP + 04]    #Recebe parâmetro
cs: 0127    F7E8      IMUL   AX               #Calcula o produto
cs: 0129    894604    MOV    [BP + 04], AX    #Prepara o valor de retorno
cs: 012C    5D        POP    BP
cs: 012D    C3        RET                     #Fim da rotina int func (parâmetro).
```

Figura 9.13(ad) Programa gerado pelo montador do DOS 5.0, relativo à montagem do programa mostrado na Fig. 9.13(d).

```
REAL     X, Y, Z, T
X:       X + Z - T
IF       X <> 0
THEN     PRINT X
ELSE     END
END
```

Figura 9.14 Programa em linguagem de alto nível para solucionar a expressão X = Y + Z − T.

b) as variáveis usadas no programa são:

Variável	Endereço (hexadecinal)	Valor (hexadecimal)
Y	1F	051
Z	20	03E
T	21	003
X	22	01A

c) o programa está armazenado na MP a partir do endereço 18h, e no instante inicial vamos considerar que:

I) CI = 18h (a letra h colocada ao lado do número indica que o valor está representado em base 16 — hexadecimal) e

II) que os valores armazenados no RI e ACC são da instrução anterior, não importante para o início de nosso programa.

O trecho da MP onde o programa e os dados estão armazenados é apresentado na tabela da Fig. 9.16 com todos os valores indicados em hexadecimal, correspondentes ao valor real armazenado em binário na memória.

Ao iniciar a execução do primeiro ciclo de instrução (a partir do endereço armazenado no CI = 18h), os valores iniciais, armazenados no RI e ACC, deverão ser destruídos pela execução desse programa (provavelmente são resultados do programa anterior).

```
              ORG
              LDA        Y
              ADD        Z
              SUB        T
              STR        X
              JZ         FIM
              PRT        X
FIM           HLT
```

Figura 9.15 Programa em linguagem de montagem para solucionar a expressão X = Y + Z − T.

A Fig. 9.17 mostra a execução do programa por meio de um quadro com os valores do CI, do RI e do ACC ao final da execução de cada uma das instruções. A primeira linha do quadro mostra a 1.ª instrução. A execução do programa é iniciada pela busca de sua primeira instrução, armazenada no endereço 18h (indicado pelo valor do CI). Essa instrução é 11F, composta do C.Op. (valor igual a 1h) e operando (valor igual a 1Fh, conforme o formato da instrução descrito na Fig. 6.19). O código de operação 1 significa: "armazenar no ACC o conteúdo da célula de endereço 1F (campo Op. da instrução)" (LDA IF). Ver descrição das instruções na Fig. 6.19.

Nessa posição (1Fh) está armazenado o valor 051h (endereço simbólico Y). Ao concluir a execução desse ciclo, o CI já estará apontando para a próxima instrução (endereço 19h).

ENDEREÇOS	CONTEÚDOS
18	11F
19	320
1A	421
1B	222
1C	51E
1D	B22
1E	000
1F	051
20	03E
21	003
22	01A

Figura 9.16 Tabela contendo trecho da MP onde se encontram armazenados o programa e os dados do exemplo da Fig. 9.15.

	CI	RI	ACC
	18	XXX	XXX
Instrução 1:	19	11F	051
Instrução 2:	1A	320	08F
Instrução 3:	1B	421	08C
Instrução 4:	1C	222	08C
Instrução 5:	1D	51E	08C
Instrução 6:	1E	B22	08C
Instrução 7:	1F	000	08C

Figura 9.17 Quadro demonstrativo de execução de um programa (programa da Fig. 9.15).

Prossegue a execução do programa, com a UC comandando a busca da segunda instrução, armazenada no endereço 19h; o valor hexadecimal é 320. O código de operação 3h corresponde à instrução ADD Op. O endereço 20h está simbolizado pela variável Z no programa da Fig. 9.15.

Na realidade, está sendo realizada a soma de Y (já no ACC) com Z, e o resultado será mantido no ACC.

A instrução seguinte, armazenada no endereço 1Ah, é 421h. O código de operação 4h (instrução SUB Op.) significa: "subtrair, do conteúdo do ACC, o conteúdo da célula de endereço 21h (simbolicamente representado por T), armazenando o resultado no ACC".

No final do ciclo dessa instrução, o CI aponta para o endereço 1Bh, e o ACC contém o resultado de Y + Z − T, que é igual ao valor em hexadecimal 08C.

Prossegue a execução do programa com o ciclo da instrução armazenada em 1Bh, cujo valor é 222h. A instrução é transferida para o RI, e o CI é incrementado de 1, armazenando então o valor 1Ch (endereço da próxima instrução).

A instrução (código de operação igual a 2h) significa: "armazenar o conteúdo do ACC na célula de MP de endereço igual ao valor do campo operando" (no nosso exemplo, o valor do campo operando é 22h). A execução da instrução consome um ciclo de escrita para gravar o valor 08Ch no referido endereço.

Inicia-se, em seguida, o ciclo da instrução armazenada no endereço 1Ch (o conteúdo da célula tem o valor hexadecimal igual a 51E), que é transferido para o RI; o CI passa a apontar para o endereço da instrução seguinte, que é 1Dh.

O código 5h (correspondente à instrução JZ Op.) significa: "desviar para o endereço 1Eh se o valor do ACC = 0". Como este valor não é igual a zero, não ocorre o desvio (o CI permanece com seu valor anterior — 1Dh). Se o valor do ACC fosse igual a zero, o valor corrente do CI (no momento é 1Dh) seria alterado para 1Eh e a instrução seguinte a ser buscada pela UC seria 000, que está armazenada no endereço 1E.

A instrução seguinte é B22h, cujo código — Bh — significa: " imprimir o valor armazenado na MP no endereço 22". O CI, no final, estará apontando para 1Eh, que passa a ser transferida para o RI, o CI é incrementado para 1Fh e, após sua decodificação, o programa termina.

Excetuando o cômputo da instrução PRT, a execução do programa consumiu 10 acessos à memória, isto é, 10 ciclos de memória. A efetiva execução da instrução PRT consome vários ciclos devido à necessária comunicação entre a UCP e o periférico (ver item 10.2).

EXERCÍCIOS

1) Explique o que você entende por compilação.

2) E por interpretação?

3) Compare os dois modos: compilação e interpretação. Indique em que circunstâncias um modo é mais vantajoso que o outro.

4) Por que um programa em linguagem Assembly não é diretamente executável pelo processador? Como este problema é, na prática, resolvido?

5) Por que um compilador deve ser específico para uma determinada linguagem de programação e para uma determinada UCP?

6) Explique o que é e como funciona o processo de ligação (linkedição).

7) Considere um computador com instruções de um operando e endereçamento por palavras de 16 bits, possuindo o seguinte conjunto de instruções:

Cod. Op. (hexadecimal)	Sigla (Assembly)	Descrição
0	END	Fim de execução
1	ADD Op.	ACC ← ACC + (Op.)
2	SUB Op.	ACC ← ACC − (Op.)
3	LDA Op.	ACC ← (Op.)
4	STA Op.	(Op.) ← ACC
5	AND Op.	ACC ← ACC and (Op.)
6	XOR Op.	ACC ← ACC xor (Op.)
A	JMP Op.	CI ← Op.
B	JP Op.	Se ACC > 0, então: CI ← Op.
C	JZ Op.	Se ACC = 0, então: CI ← Op.
D	JN Op.	Se ACC < 0, então: CI ← Op.
E	GET Op.	Ler dado para (Op.)
F	PRT Op.	Imprimir (Op.)

Num dado instante, foi carregado um programa na memória. Os registradores da UCP têm os seguintes valores, em hexadecimal: CI = 1AF; RI = 20A3; ACC = 153C; e a fila de dados de entrada tem os valores decimais: 19, 37, 13 e 52.

End.	Conteúdo
1AF	E1C0
1B0	E1C1
1B1	31C1
1B2	11C0
1B3	41C1
1B4	D1BA
1B5	E1C2

End.	Conteúdo
1B6	31C1
1B7	21C2
1B8	41C1
1B9	A1BE
1BA	E1C3
1BB	31C1
1BC	11C3

End.	Conteúdo
1BD	41C1
1BE	F1C1
1BF	0000
1C0	31A5
1C1	61C4
1C2	21C0
1C3	11C4

Pergunta-se:

a) Qual o valor, em hexadecimal, de RI, CI e ACC ao final da execução de cada instrução?

b) Quais os valores impressos em decimal?

c) O que aconteceria com o programa se o conteúdo da posição de endereço 1B3 fosse alterado para 81C1?

8) Qual é a diferença entre um código-objeto e módulo de carga? Em que eles são comuns?

9) O que você entende pelo termo "referência externa" em um programa?

10) Conceitue a técnica de desvio na execução de um programa. Quais são os tipos de desvio que podem ser implementados por instruções de máquina? Dê exemplos de comandos em linguagens de alto nível que implementam desvios.

11) Quais são as principais etapas de análise de um programa-fonte realizadas por um programa compilador?

12) Supondo que um determinado algoritmo foi codificado por um programador em linguagem Pascal e compilado em um microcomputador do tipo PC (possui um processador Intel 80486), gerando um determinado código, é possível executar este código, gerado pelo referido compilador, em um microcomputador Macintosh, da Apple (possui um processador Motorola 68030)? Por quê?

10

Entrada e Saída (E/S)

10.1 INTRODUÇÃO

No Cap. 2 apresentamos de modo resumido os componentes básicos de um sistema de computação: a Unidade Central de Processamento — UCP, a Memória Principal — MP, a Memória Secundária — MS e os módulos de Entrada/Saída — E/S. Nos capítulos subseqüentes, descrevemos como a UCP e a MP funcionam, de modo individual e interligado, na tarefa de executar um programa. A interligação UCP/MP pode ser vista como mostrado na Fig. 10.1, através do triplo barramento — de dados, endereço e sinais de controle.

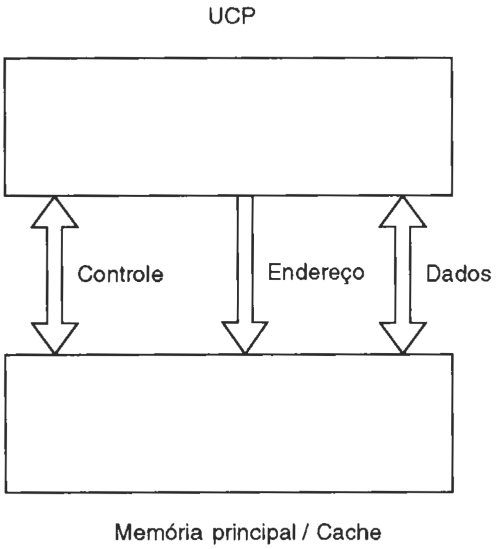


Figura 10.1 Interligação UCP/MP com barramento triplo.

No entanto, para que possamos desfrutar da rapidez e flexibilidade de um computador, não basta sabermos que ele pode armazenar na memória os programas e dados que desejamos processar, nem que ele pode executar mais de um milhão de instruções por segundo. É preciso que o programa que temos escrito em uma folha de papel e os dados que serão por ele manipulados sejam inseridos no sistema, caractere por caractere, inclusive os espaços em branco entre os caracteres, os sinais de pontuação e os símbolos de operações matemáticas. Para tanto, precisamos de um meio qualquer que faça essa comunicação homem-máquina. Um teclado do tipo semelhante ao de uma máquina de escrever pode servir como elemento de entrada.

Em geral, os dispositivos de entrada ou de saída são denominados *periféricos* (porque se encontram instalados fora do núcleo principal UCP/MP, mas ficam na maior parte das vezes próximos, isto é, na sua periferia). A Fig. 10.2 mostra um esquema representativo da comunicação entre o usuário e a máquina, para introduzir informações no sistema.

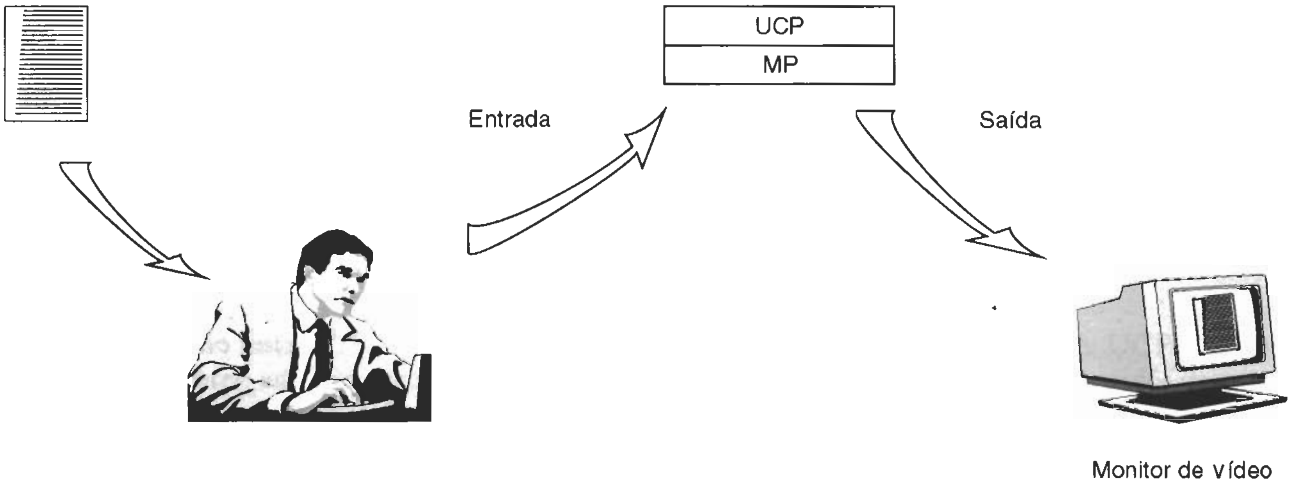


Figura 10.2 Exemplo de comunicação homem-máquina.

Da mesma forma que temos a necessidade de comunicação com a máquina, também é preciso que haja comunicação no sentido contrário, isto é, máquina-homem, de modo que o usuário possa entender os resultados de um processamento. Uma impressora ou uma tela de vídeo pode servir como dispositivo de saída ou periférico de saída (no decorrer deste capítulo usaremos os dois termos indistintamente, dispositivo de entrada ou de saída ou simplesmente periférico). A mesma Fig. 10.2 mostra este tipo de comunicação.

A Fig. 10.1 mostra o modo de interligação adotado para a comunicação UCP/MP: o *barramento*. Este mesmo método define a interligação do conjunto UCP/MP aos periféricos. É através do barramento do sistema (*system bus*), já citado no Cap. 6, que se pode, então, interligar todos os componentes de um sistema de computação e por onde fluem os mesmos tipos de informação, dados, endereços e sinais de controle. A Fig. 10.3 mostra um diagrama simplificado dessas ligações.

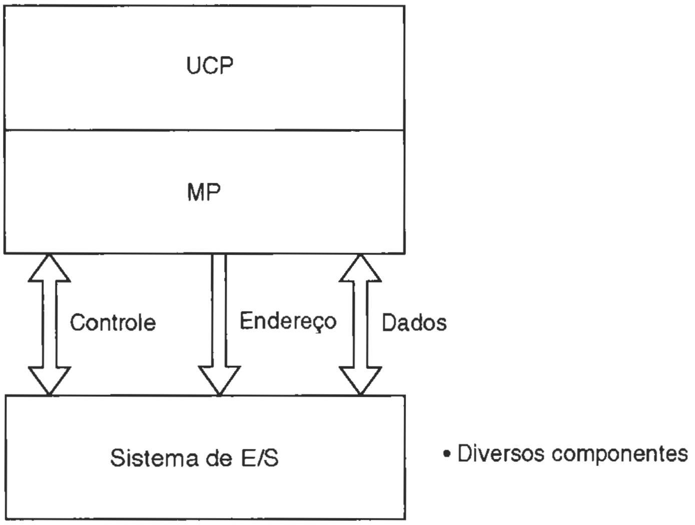


Figura 10.3 Interligação do conjunto UCP/MP com o conjunto da E/S através de um barramento de dados, de endereços e de sinais de controle.

Na realidade, o barramento do sistema permite o compartilhamento de informações entre os diversos componentes de um computador, da mesma forma que o barramento interno da UCP permite o trânsito de informações entre os registradores e demais unidades da UCP. A Fig. 10.4 mostra dois exemplos do uso do barramento do sistema em computadores, conforme já mostrado no item 6.6.3. Na Fig. 10.4(a) é apresentado um esquema de barramento múltiplo, com canais separados para dados, endereços e sinais de controle, enquanto, na Fig. 10.4(b), é mostrado um exemplo de barramento único (unibus).

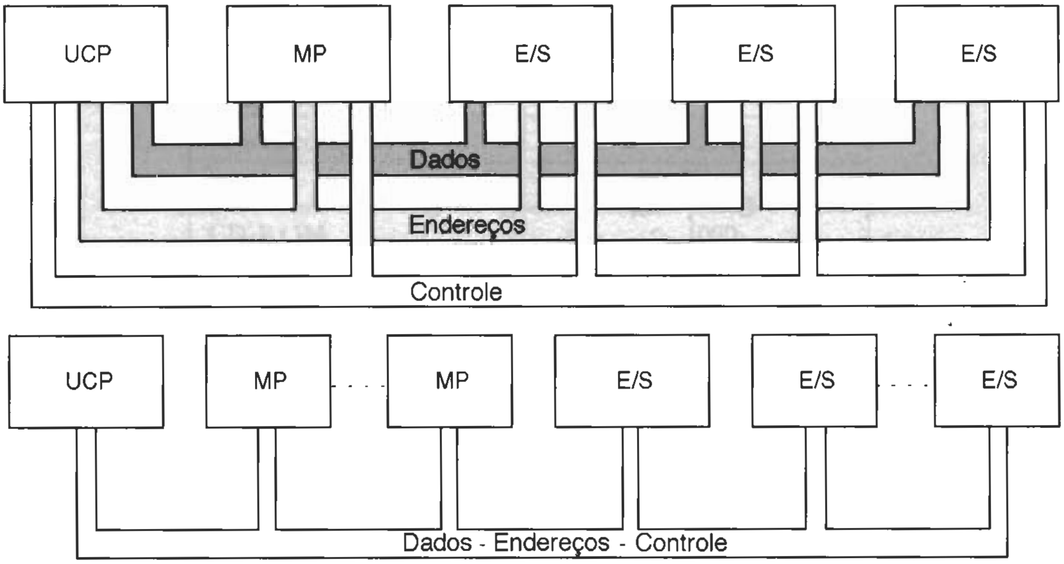


Figura 10.4(a) Barramento múltiplo (b) Barramento único.

Conforme pudemos observar no Cap. 6, há diversos tipos de barramento utilizados nos atuais sistemas de computação, os quais podem ser genericamente classificados em três categorias:

- barramento local;
- barramento do sistema;
- barramento de expansão.

Os dois primeiros estão relacionados à interligação do processador com os módulos de memória cache e principal, enquanto o barramento de expansão foi a maneira encontrada para interligar os periféricos, dispositivos de velocidade mais baixa que a UCP/Memória, aos elementos de maior velocidade (UCP/MP) (ver Fig. 6.40).

Também vimos no Cap. 6 que o barramento de expansão pode ser implementado em duas partes, uma para os dispositivos de E/S de maior velocidade, como os discos, modems, rede, e a outra, mais adequada aos dispositivos de menor velocidade, como teclado, mouse e outros.

O funcionamento do conjunto de dispositivos de entrada/saída em um computador é, em geral, caracterizado pela existência de diversos elementos que, embora realizem o mesmo tipo de função (tenham o mesmo objetivo de comunicação homem-máquina), possuem características bem diversas. Por isso, costuma-se integrar os diversos elementos que cooperam no processo de entrada e saída, em um subsistema, como parte do sistema de computação.

Um subsistema de entrada/saída (E/S) deve, em conjunto, ser capaz de realizar duas funções:

– receber ou enviar informações ao meio exterior;

– converter as informações (de entrada ou de saída) em uma forma inteligível para a máquina (se estiver recebendo) ou para o programador (se estiver enviando).

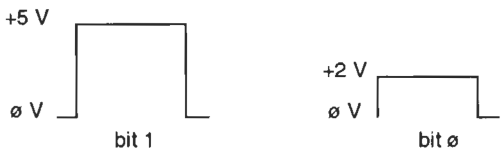
A Fig. 10.5(a) mostra exemplos de símbolos utilizados pelos seres humanos para representar informações, enquanto a Fig. 10.5(b) mostra os "símbolos" usados pelos computadores. No primeiro caso, a grande variedade de formas dos símbolos é interpretada pelo ser humano primariamente por observação visual (o leitor está interpretando as informações contidas neste livro através da visualização dos sinais gráficos que representam os diversos símbolos — caracteres alfabéticos, algarismos decimais, sinais de pontuação, etc.).

No caso dos computadores, a variedade se restringe apenas aos valores 0 e 1 e, por serem máquinas, não utilizam o processo visual entendendo dois diferentes níveis de intensidade de sinais elétricos, campos magnéticos com dois sentidos de magnetização, etc. Daí termos mencionado a função *"converter"* dos dispositivos ou periféricos de E/S. Eles convertem, por exemplo, o movimento de pressão de uma tecla em vários sinais elétricos, com intensidades diferentes (conforme o valor desejado seja 0 ou 1).

A ; a (+ @ &

5 , x | û ë -

(a) Símbolos usados nas linguagens do ser humano



(b) Símbolos usados na linguagem dos computadores

Figura 10.5 Símbolos utilizados por pessoas (a) e por computadores (b).

Sobre a relação e comunicação entre o conjunto UCP/MP e o subsistema E/S há algumas observações interessantes que devem ser mencionadas nesse ponto:

a) A primeira observação refere-se às diferentes características de cada dispositivo de E/S, o que tornaria extremamente complicada a comunicação UCP → periférico, se esta fosse realizada direta e individualmente, isto é, se houvesse uma comunicação direta entre a UCP e o teclado, entre a UCP e a impressora, entre a UCP e o vídeo, e assim por diante. A Fig. 10.6 mostra um esquema desse tipo de comunicação, apenas para o entendimento do leitor, já que ele não é prático nem economicamente viável. Da figura, podemos observar que o teclado é um dispositivo lento comparativamente com os discos e que o mouse e o teclado enviam os bits um a um, ao passo que o vídeo e a impressora recebem da UCP as informações byte a byte. Já os discos e disquetes trocam informações com o conjunto UCP/MP em blocos de bits para otimizar a transferência. Obviamente, nessa figura não estão assinaladas todas as diferenças entre os diversos dispositivos, existindo diferenças até mesmo relativas à parte elétrica de geração e interpretação dos sinais de transmissão.

A Tabela 10.1 apresenta uma relação de dispositivos de E/S e a velocidade média com que transferem informações para o interior do sistema, de modo que possamos entender melhor uma das diferenças entre eles.

Devido a essas diferenças, na prática a UCP não se conecta diretamente com cada periférico, mas sim com dispositivos que realizam a "tradução" e a compatibilização das características de um (dispositivo de E/S) para o outro (UCP/MP), além de realizar outras tarefas de controle. Esses dispositivos costumam ser chamados de interface de E/S, porém há outros nomes igualmente utilizados pelo mercado, por exemplo, controlador (inserindo-se também o nome específico do periférico, como controlador de disco, controlador de vídeo), processador de periféricos, canal, adaptador e outros. Mas a função de todos é sempre a mesma: compatibilizar as diferentes características de um periférico e da UCP/MP, permitindo um fluxo correto de dados em uma velocidade adequada a ambos os elementos que

Tabela 10.1 Exemplo de Dispositivos de E/S e a Sua Velocidade de Transmissão de Dados

Dispositivo	**Taxa de transmissão (KB/s)**
Teclado	0,01
Mouse	0,02
Impresssora matricial	1
Modem	2 a 8
Disquete	100
Impressora laser	200
Scaner	400
CD-ROM	1000
Rede local	500 a 6000
Vídeo gráfico	60.000
Disco rígido (HD)	2000 a 10.000

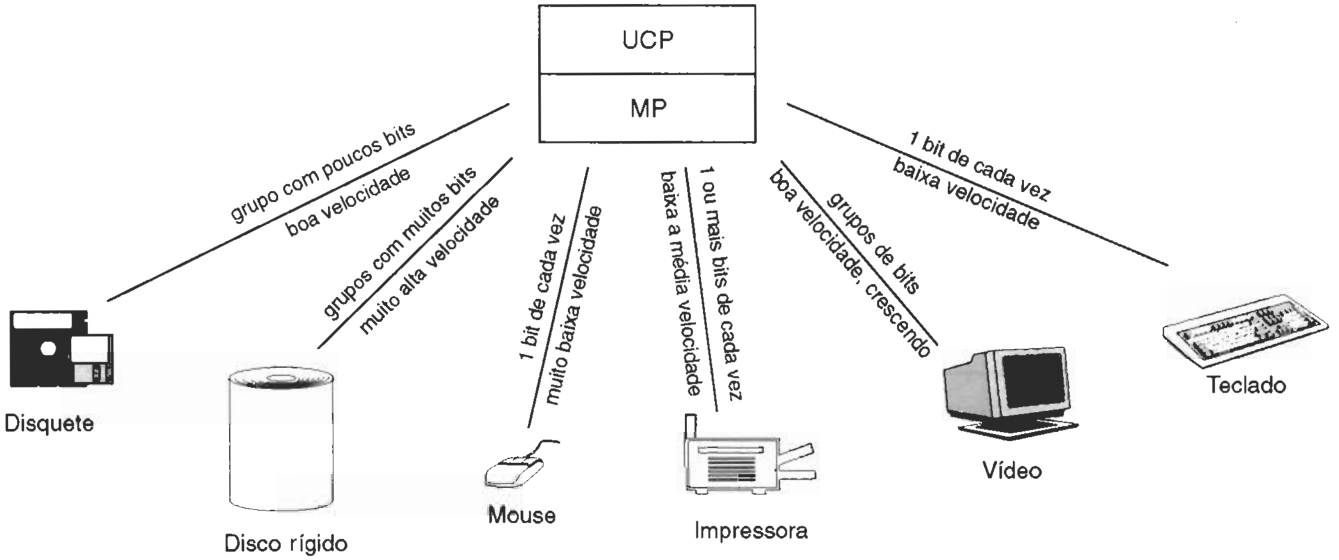

UCP
MP
grupo com poucos bits
boa velocidade
grupos com muitos bits
muito alta velocidade
1 bit de cada vez
muito baixa velocidade
1 ou mais bits de cada vez
baixa a média velocidade
grupos de bits
boa velocidade, crescendo
1 bit de cada vez
baixa velocidade
Disquete
Disco rígido
Mouse
Impressora
Vídeo
Teclado

Figura 10.6 Exemplo de comunicação direta UCP/MP e periféricos, indicando-se as diferentes características de transmissão de cada um.

estão sendo interconectados. No item 10.2 serão apresentados apenas os aspectos mais relevantes sobre interfaces de E/S e, no item 10.5, o assunto será visto com mais detalhe.

b) Os diversos tipos de dispositivos que podem ser conectados em um computador são classificados em três categorias:

1) os que transmitem/recebem informações inteligíveis para o ser humano — são adequados para estabelecimento de comunicação com o usuário. É o caso de impressoras, monitores de vídeo, teclados;

2) os que transmitem/recebem informações inteligíveis apenas para a máquina — são adequados para comunicação máquina a máquina ou internamente a uma máquina. Exemplos desta categoria são os discos magnéticos e sensores;

3) os que *transmitem/recebem de/para outros dispositivos remotamente instalados, tais como os modems* e regeneradores digitais em redes de comunicação de dados.

c) Há duas maneiras básicas de se realizar transmissão/recepção de dados entre os periféricos/interfaces e UCP/MP, bem como entre dispositivos interconectados entre si, local ou remotamente:

   1) a informação pode ser transmitida/recebida, bit a bit, um em seguida ao outro — isto caracteriza a *transmissão serial*; e

   2) a informação pode ser transmitida/recebida em grupos de bits de cada vez, isto é, um grupo de bits é transmitido simultaneamente de cada vez. Chama-se *transmissão paralela.*

   A escolha de um desses tipos para interligar os elementos de E/S ao sistema UCP/MP depende de vários fatores, tais como: tipo e natureza do periférico, custo de implementação e velocidade de transmissão desejada.

d) Basicamente, um dispositivo de entrada ou saída se comunica com o meio externo (usuário do sistema ou outro dispositivo) e com seu interface de E/S. Esta comunicação compreende o envio e recebimento de dados (bits) e sinais de controle. A Fig. 10.7 mostra os tipos de informação, bem como o sentido da direção do fluxo, transmitidas/recebidas entre os elementos que se conectam a um dispositivo periférico. Embora cada dispositivo tenha características de funcionamento próprias e distintas dos outros, o fluxo de informações é basicamente o mesmo.

   Na figura, podemos identificar três linhas de transmissão de informações entre o dispositivo e seu interface. Uma consiste na interligação entre o próprio dispositivo e o mundo externo, para transferência de dados entre ambos. A comunicação de controle entre o interface e o dispositivo se realiza através das duas outras vias: uma, de sinais de controle, cuja direção do fluxo é do interface para o dispositivo, onde passam os sinais enviados pelo interface para o dispositivo, do tipo: informe seu estado; solicito uma leitura (Read) de dados; solicito uma escrita (Write) de dados. A outra via, de sinais enviados do dispositivo para o interface, do tipo: meu estado é pronto (Ready) ou ocupado (Busy ou Not Ready).

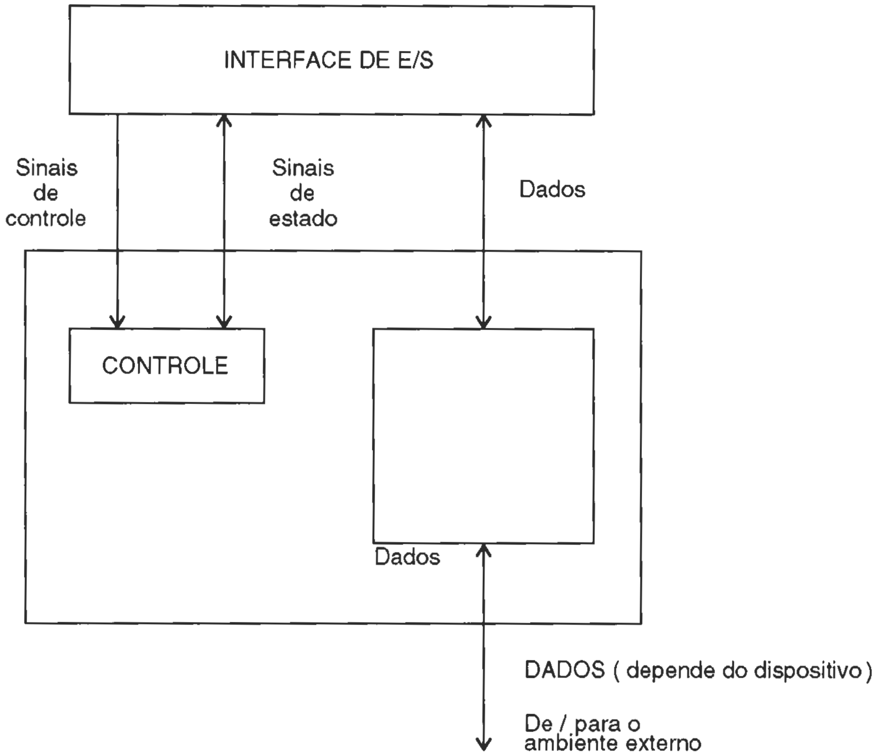


Figura 10.7 Esquema de funcionamento entre um dispositivo de E/S e seu interface.

Resumindo as observações efetuadas, vamos descrever um pouco mais os aspectos concernentes aos seguintes tópicos:

- Sobre os interfaces de E/S, isto é, sobre os dispositivos que interligam os periféricos ao sistema central UCP/MP.
- Sobre as modalidades de interligação entre os elementos do sistema, ou seja: sobre a transmissão tipo serial e sobre a transmissão tipo paralela.
- Sobre alguns dispositivos de E/S, isto é, vamos analisar equipamentos como o teclado, o vídeo, a impressora, o disco magnético, o CD-ROM.

10.2 INTERFACES DE E/S

No item anterior já verificamos a necessidade de um elemento entre o conjunto UCP/MP e um periférico, visando a compatibilizar as diferentes características entre ambos — o *interface*. Vamos repetir este ponto, enfatizando os aspectos essenciais:

a) Há no mercado uma quantidade muito grande de dispositivos periféricos que podem ser conectados a uma UCP, cada um possuindo modos próprios e específicos de funcionar, certamente diferentes uns dos outros, bem como diferentes entre cada um deles e a UCP (ver Fig. 10.6). Seria impraticável dotar a UCP de lógica específica para tratar com cada periférico.

b) A velocidade de transferência de dados de um periférico é, em geral, muito menor que a da UCP (ver Tabela 10.1). Não seria eficaz conectar os periféricos diretamente ao barramento do sistema, pois isto reduziria a velocidade da UCP/MP em sua comunicação. Além disso, cada periférico também tem velocidade diversa de outro periférico, o que corrobora ainda mais a afirmação de que não é possível usar o mesmo caminho por usuários diversos. Por exemplo, o teclado é muito mais lento que um disco, que, por sua vez, é muito mais lento que a MP.

c) Os periféricos costumam usar formatos e tamanhos diferentes de *unidades de transferência de dados*, isto é, uns transferem 1 bit de cada vez (a unidade de transferência é, então, o bit), outros transferem um byte ou caractere por vez, enquanto alguns periféricos podem enviar ou receber centenas ou até mesmo milhares de bits em um único bloco de transferência. Além disso, os fabricantes também adotam diversos tamanhos de *palavra de dados* da UCP (o processador Pentium, da Intel foi especificado com palavra de 16 bits, enquanto o processador Alpha (RISC) da DEC possui palavra de 64 bits, porém ambos podem usar o mesmo teclado).

Esses aspectos confirmam, portanto, a necessidade do emprego de um dispositivo intermediário, ligando a UCP/MP a um periférico ou a um grupo de periféricos, o qual costuma ser chamado de Interface de E/S, mas que, conforme já mencionamos no item anterior, também pode ser identificado com vários outros nomes, dependendo do fabricante ou do mercado. Por exemplo, a IBM criou o nome de *channel* (canal), enquanto os fabricantes de microcomputadores costumam chamar de *adapter* (adaptadores) ou controladores, e assim por diante. Popularmente, também denominamos um adaptador de "placa", visto consistir realmente em uma placa de circuito impresso. Assim, adquirimos no mercado uma placa de vídeo (interface para o monitor de vídeo) ou uma placa fax/modem (interface para os mencionados dispositivos).

Um interface de E/S pode servir apenas para a conexão à UCP/MP e para controle de um único dispositivo de E/S ou pode atender a vários dispositivos, até mesmo dispositivos diferentes. Um controlador de disco de certos microcomputadores pode controlar a comunicação da UCP/MP com uma unidade de disco e uma unidade de disquete. Há outros tipos de controladores, em microcomputadores, que podem controlar até 8 dispositivos periféricos (Interface SCSI — Small Computer System Interface). A Fig. 10.8 mostra alguns exemplos de conexões de UCP/MP a controladores de E/S ou interfaces.

Um interface ou controlador de E/S é, em geral, responsável pelas seguintes tarefas:

a) controlar e sincronizar o fluxo de dados entre a UCP/MP e o periférico;

b) realizar a comunicação com a UCP, inclusive interpretando suas instruções ou sinais de controle para o acesso físico ao periférico;

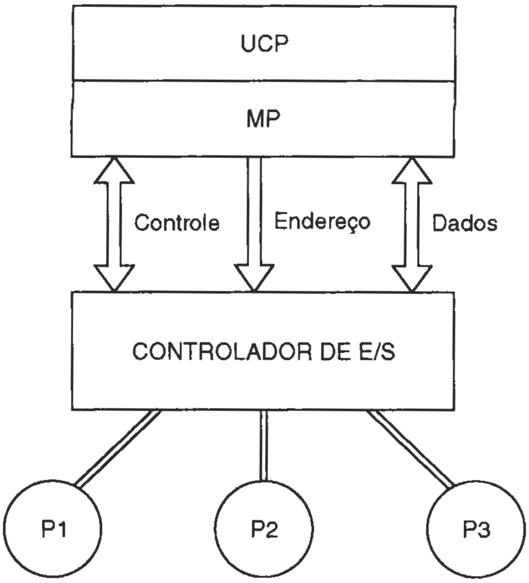

UCP
MP
Controle
Endereço
Dados
CONTROLADOR DE E/S
P1
P2
P3

Figura 10.8(a) Exemplo de configuração UCP/MP e um interface de E/S.

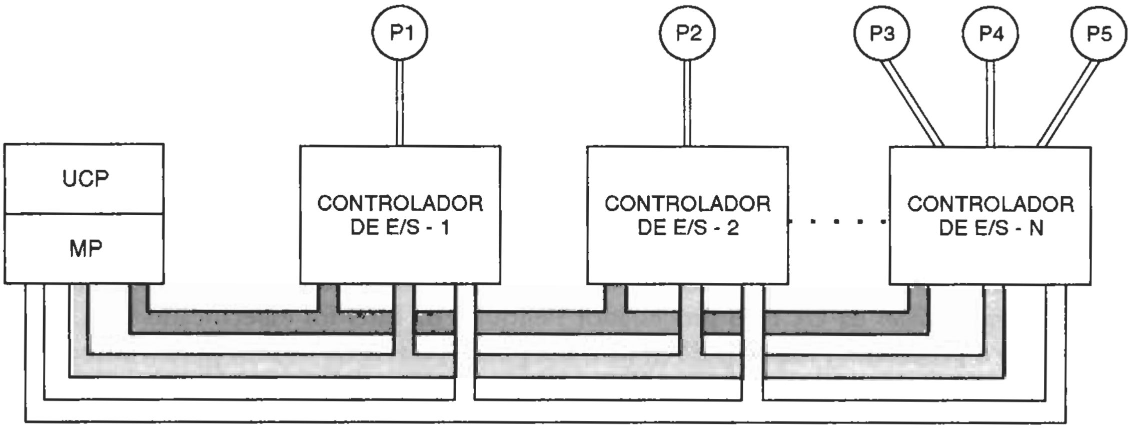

P1
P2
P3
P4
P5
UCP
MP
CONTROLADOR DE E/S - 1
CONTROLADOR DE E/S - 2
CONTROLADOR DE E/S - N

Figura 10.8(b) Exemplo de configuração UCP/MP e um interface.

c) servir de memória auxiliar para o trânsito das informações entre os componentes (buffer de dados); e

d) realizar algum tipo de detecção e correção de erros durante as transmissões (ver item 5.4).

Em geral, um interface de E/S se comunica com o processador e com o periférico através das seguintes ações básicas:

1) o processador interroga o interface para verificar se o periférico está pronto para se comunicar ou se está ocupado;

2) o interface responde informando o estado do periférico; se está pronto (Ready) ou ocupado ou desligado ou outra incapacidade qualquer de atender (Not Ready);

3) se o dispositivo de E/S (periférico) estiver pronto para se comunicar (receber ou enviar dados), então o processador envia os dados pelo barramento para o interface. Deste, os dados irão para o dispositivo ou vice-versa.

A utilização de um buffer interno pelo interface é um fator fundamental para a compatibilização de velocidades diferentes entre o barramento do sistema e suas linhas externas.

Uma transmissão do periférico ou do controlador pode ser realizada:

- *bit a bit* e, neste caso, temos *transmissão serial*; ou
- *em grupos de bits de cada vez* e, neste caso, temos *transmissão em paralelo.*

Vejamos como estes dois tipos de transmissão funcionam.

10.2.1 Transmissão Serial

Na *transmissão serial*, o periférico é conectado ao dispositivo controlador ou interface de E/S por uma única linha de transmissão de dados, de modo que a transferência de dados é realizada um bit de cada vez, embora o controlador possa ser conectado à UCP/MP através de barramento com várias linhas, conforme mostrado na Fig. 10.9.

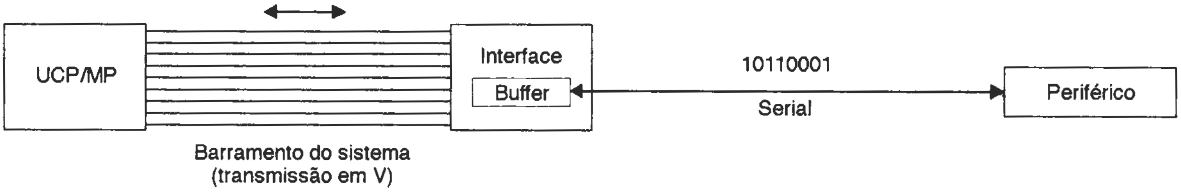


Figura 10.9 Exemplo de transmissão serial (interface-periférica).

A transmissão serial é mais lenta que a transmissão paralela, visto que só envia um bit de cada vez, sendo normalmente utilizada em periférico de baixa velocidade ou cuja característica é típica de transmissão bit a bit. O *teclado* e o *mouse* são dispositivos que realizam comunicação serial, assim como os *modems* (equipamentos utilizados para enviar dados, via linhas telefônicas, para outros dispositivos geograficamente distantes).

Como a transmissão é realizada bit a bit, é necessário que o receptor e o transmissor estejam sincronizados bit a bit, isto é, o transmissor transmite os bits sempre com a mesma velocidade e, conseqüentemente, todos os bits terão a mesma duração no tempo. Por exemplo, se o transmissor estiver funcionando na velocidade de 1000 bits por segundo (abrevia-se para 1000 bps), isto significa que cada bit dura 1/1000 segundos ou 1 milissegundo.

Para que o receptor seja capaz de receber todos os bits (um por um) enviados, ele precisa saber quando um bit inicia e qual é a sua duração (também se costuma chamar de largura do bit). Se, a cada 1 ms, o transmissor envia um bit (o nível de tensão alto significa, por exemplo, bit 1 e o nível de tensão baixo significa bit 0), então, a cada 1 ms o receptor deve "sensar" o nível de tensão da linha para captar o bit que está chegando e identificá-lo como 0 ou 1.

O receptor deve trabalhar, para isso, com a mesma velocidade (1000 bps, no exemplo) do transmissor. A Fig. 10.10 mostra um exemplo desse processo de sincronização de bits. Este processo é eficaz para a identificação de cada bit, porém ainda não é suficiente para a identificação de um caractere, já que é preciso definir quando um caractere inicia (qual é seu primeiro bit) e quando ele termina, ou seja, deve-se criar um método de identificação do bit inicial do caractere.

Conforme observado na Fig. 10.10, transmissor e receptor estão funcionando na mesma velocidade de 1000 bps, o que implica a geração de bits com duração igual a 1 ms. Dessa maneira, o receptor deve *"sensar"* a linha a cada 1 ms e captar o bit adequado, de acordo com o nível de tensão que for *"sensado"*. Para que haja maior confiabilidade no processo, é comum que o receptor *"sense"* no instante em que o bit está na metade de sua duração, de modo a evitar possíveis erros se ele, por exemplo, detectasse próximo à subida do valor 0 para 1 ou vice-versa.

Receber bit por bit não é suficiente. É preciso que o receptor saiba identificar grupos de bits que tenham um significado, como, por exemplo, o da representação de 1 caractere. Dependendo do código de represen-

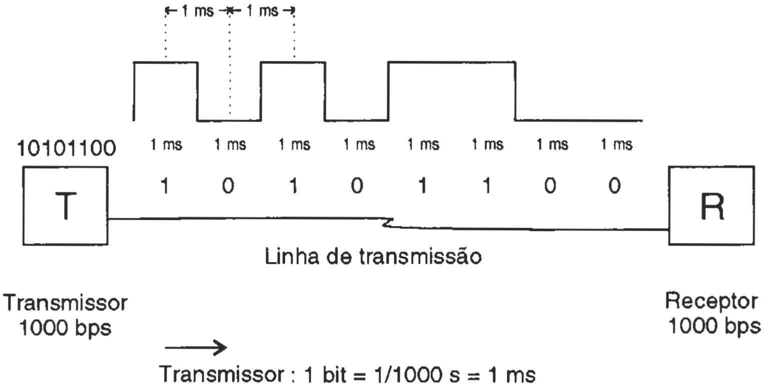

1 ms
1 ms
10101100
1 ms
1 ms
1 ms
1 ms
1 ms
1 ms
1 ms
1 ms
1
0
1
0
1
1
0
0
T
R
Linha de transmissão
Transmissor
1000 bps
Receptor
1000 bps
Transmissor : 1 bit = 1/1000 s = 1 ms

Figura 10.10 Exemplo de sincronização em nível de bit.

tação utilizado (ver item 7.2), cada caractere será representado por um grupo de *n* bits (em geral, *n* é igual a 8 bits, que é o mesmo valor de 1 byte).

Há dois métodos de se realizar transmissão serial:

- *transmissão assíncrona*; e
- *transmissão síncrona*.

10.2.1.1 Transmissão Assíncrona

É o método mais antigo, simples e barato, utilizado por antigos terminais TTY e que sempre foi usado em larga escala por microcomputadores. Ele consiste em um processo de sincronização do receptor a cada novo caractere transmitido (daí o nome assíncrono). Para isso, antes de se iniciar a transmissão, cada caractere é acrescido de 2 pulsos, um no início do caractere, denominado START, com a duração exata de 1 bit e valor de tensão correspondente ao bit 0, e o outro, denominado STOP, tem valor de tensão igual ao do bit 1 e duração variável entre 1 e 2 bits. No caso do START, trata-se de um bit 0 inserido antes do primeiro bit do caractere, passando a ser o novo primeiro bit do caractere, conforme mostrado na Fig. 10.11.

A Fig. 10.11 mostra um caractere ASCII transmitido pelo método assíncrono, isto é, os 8 bits ASCII, mais o START no início e o STOP no final do caractere. Quando não há transmissão, o transmissor envia continuamente bits 1 pela linha (nível alto de tensão). Quando um caractere é enviado, o receptor detecta a queda de tensão (nível alto quando não há transmissão e nível baixo do START) e entra em sincronismo, recebendo, daí para diante, os demais bits do caractere, até o STOP (ele possui um circuito contador e sabe quantos bits cada caractere tem, já que transmissor e receptor funcionam com o mesmo código de armazenamento).

Para se realizar a transmissão serial de forma assíncrona é necessário que, em ambos os lados da linha de transmissão, haja um dispositivo capaz de decompor cada caractere bit a bit e providenciar a inclusão dos bits

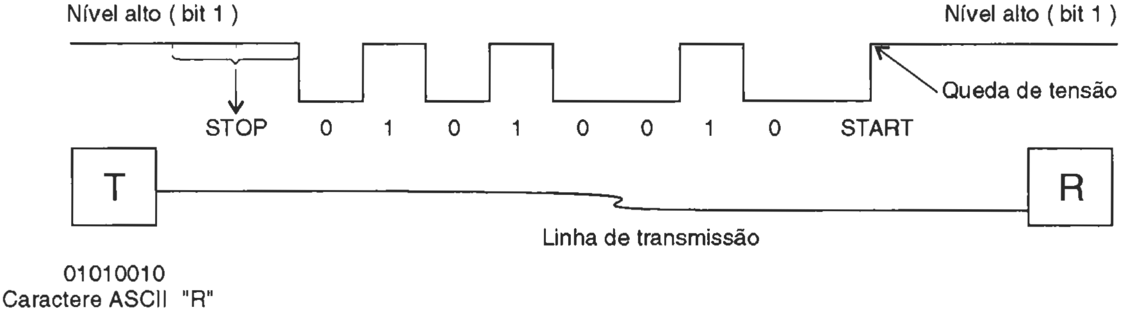

Nível alto (bit 1)
Nível alto (bit 1)
Queda de tensão
STOP
0
1
0
1
0
0
1
0
START
T
R
Linha de transmissão
01010010
Caractere ASCII "R"

Figura 10.11 Exemplo de transmissão assíncrona do caractere ASCII "R".

START/STOP na transmissão e sua retirada após a recepção do caractere. Um dispositivo muito comum, usado em microcomputadores e que faz parte da maioria das pastilhas (chips) de entrada/saída denomina-se *UART (Universal Asynchronous Receiver/Transmitter)*, transmissor/receptor universal assíncrono. A UART é uma pastilha que emprega integração em larga escala (LSI), cuja função básica é a decomposição e composição de um caractere em bits e vice-versa, conforme mostrado esquematicamente na Fig. 10.12.

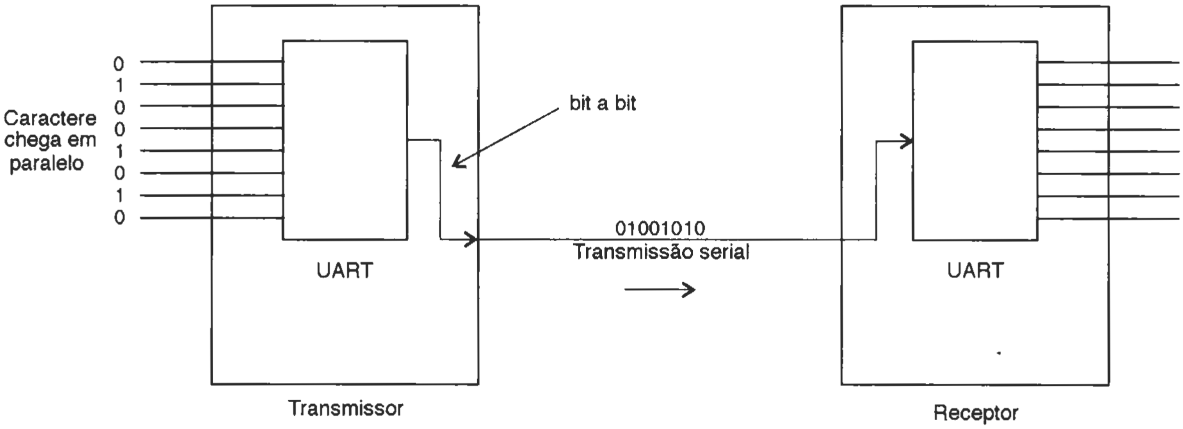

Caractere chega em paralelo
0
1
0
0
1
0
1
0
bit a bit
UART
01001010
Transmissão serial
UART
Transmissor
Receptor

Figura 10.12 Exemplo de conversão paralela/serial, transmissão serial e conversão serial/paralela com emprego de uma UART.

A Fig. 10.13 mostra o diagrama em blocos de uma UART, indicando seus principais componentes:

Buffer de saída de dados a serem transmitidos — recebe os *n* bits do caractere (podem ser 5, 6, 7 ou 8 bits, dependendo do dispositivo que está sendo usado) e os envia para o registrador de transmissão.

Registrador de transmissão — desloca os bits do caractere um a um (*shift register*) para a linha de saída. Este deslocamento é realizado a cada pulso de relógio da UART.

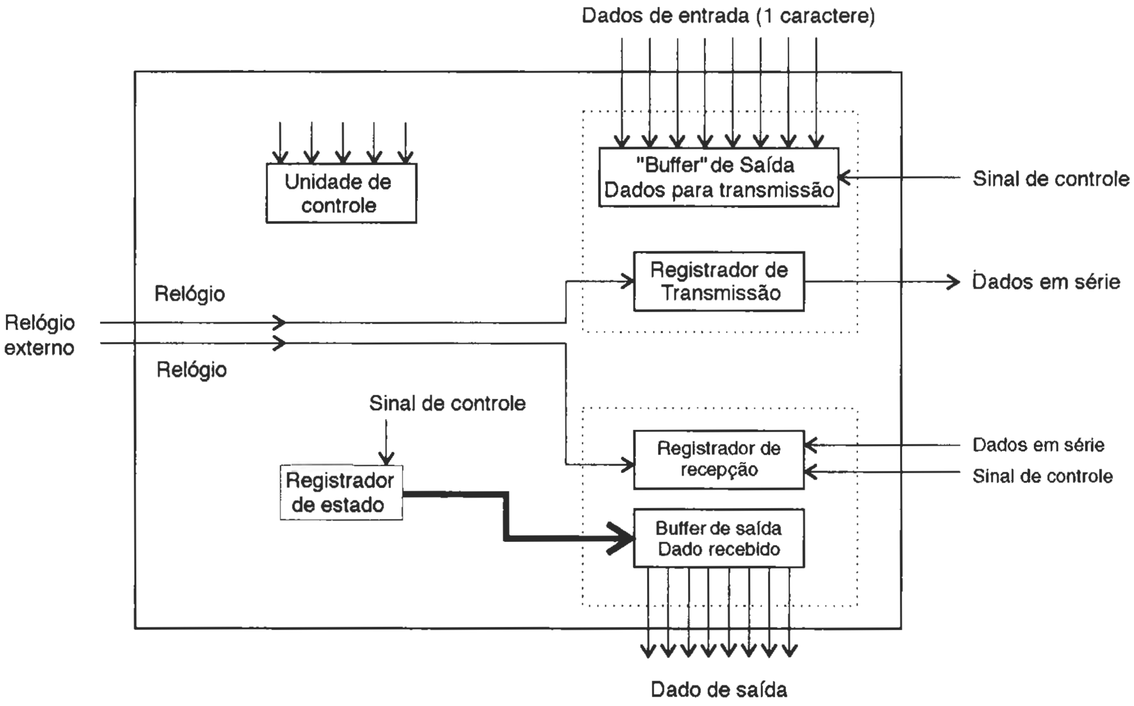

Dados de entrada (1 caractere)
Unidade de controle
"Buffer" de Saída Dados para transmissão
Sinal de controle
Registrador de Transmissão
Dados em série
Relógio
Relógio externo
Relógio
Sinal de controle
Registrador de recepção
Dados em série
Sinal de controle
Registrador de estado
Buffer de saída Dado recebido
Dado de saída

Figura 10.13 Diagrama em bloco de uma UART.

Registrador de recepção e bufffer de saída de dados recebidos — funcionam de modo semelhante, porém em sentido inverso ao dos dois registradores já descritos. O caractere é recebido bit a bit no registrador de recepção, que efetua o deslocamento de cada bit até completar todo o caractere e, então, o encaminha para o buffer de saída.

Unidade de controle — permite que a UART funcione de modo diferente, conforme a escolha do usuário: opção de paridade, e se houver, se será par ou ímpar (ver item 5.4), opção de 1 ou 2 bits STOP.

Registrador de estado (semelhante ao flag dos microprocessadores) — possui um bit para indicar algumas ocorrências durante o funcionamento da UART, tais como: erro de paridade, erro de sincronização (a UART perdeu o bit START), dados disponíveis (para que o microprocessador leia o caractere).

Relógio — divide a freqüência de transmissão (taxa de bands) para permitir o deslocamento de cada bit dos registradores e deslocamento.

Em geral, a taxa de transmissão (velocidade de sinalização é o termo mais empregado na área de teleprocessamento) é medida em *bauds* (quantidade de símbolos transmitidos por segundo), que pode variar entre 110 bauds e 38.400 bauds, sendo comuns: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19.200 e 38.400 bauds.

10.2.1.2 Transmissão Síncrona

É uma técnica mais eficiente que a transmissão assíncrona, pois são transmitidos de cada vez blocos de caracteres, sem intervalo entre eles e sem pulso START/STOP (isto reduz a quantidade de bits, que não são usados para efetiva representação dos caracteres e que ocupam a capacidade da linha). Por exemplo, uma transmissão de 100 caracteres ASCII de modo assíncrono tem uma eficiência de:

$$\frac{\text{Quantidade de informação}}{\text{Quantidade total de bits de transmissão}} = \frac{7 \text{ bits} \times 100}{(\underbrace{7}_{\text{bits de informação}} + \underbrace{1}_{\text{bit start}} + \underbrace{1}_{\text{bit stop}} + \underbrace{1}_{\text{bit de paridade}}) \times 100}$$

A eficiência de 70%, nesse caso em que todos os caracteres estão sendo considerados sem intervalo, é a mesma para a transmissão de 1 ou de N caracteres, mas poderia ser menor ainda se ocorresse intervalo entre os caracteres, pois o denominador de fração iria aumentar (um intervalo de 20 milissegundos em uma transmissão com a taxa de 100 bps corresponde a cerca de 20 bits: 20 × 1/1000).

Na transmissão síncrona, a eficiência seria:

$$E = \frac{100 \text{ caracteres}}{105 \text{ caracteres}} = 95\%$$

Os 105 caracteres do denominador compreendem os 100 caracteres da informação que se deseja transmitir mais 5 caracteres especiais necessários ao controle da transmissão e formato do bloco de caracteres (não faz parte do escopo deste livro detalhar um protocolo de comunicação de dados como o BSC ou HDLC/SDLC, onde se poderia melhor descrever a função de cada caractere especial).

A Fig. 10.14 mostra um esquema de transmissão síncrona, cujas características principais são:

a) não há intervalo entre os caracteres de um bloco, isto é, o transmissor monta um bloco, usualmente com cerca de 128 a 256 caracteres, e este é transmitido bit a bit sem intervalo entre o primeiro e o último bit; e
b) para que o receptor se mantenha sincronizado ("sensar" a linha no mesmo intervalo de tempo que dura um bit — ver Fig. 10.10) é necessário que ele funcione com a mesma freqüência do relógio do transmis-

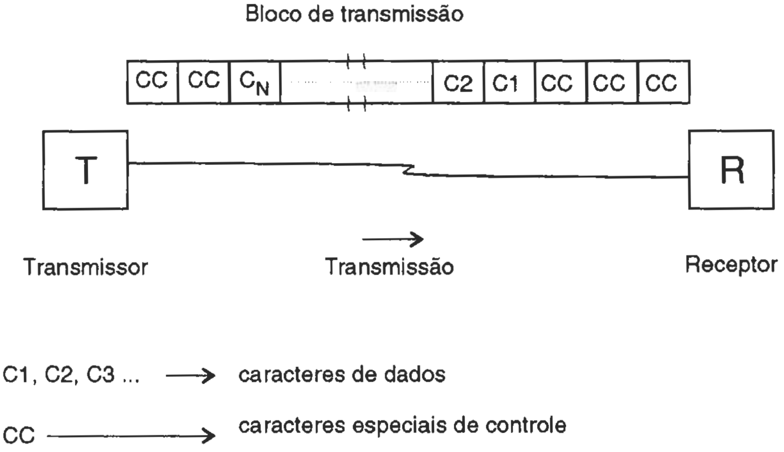

Bloco de transmissão
CC
CC
C_N
C2
C1
CC
CC
CC
T
R
Transmissor
Transmissão
Receptor
C1, C2, C3 ... → caracteres de dados
CC → caracteres especiais de controle

Figura 10.14 Exemplo de transmissão síncrona.

sor. Uma possibilidade de sincronizar os relógios é incluir uma linha de transmissão separada por onde circulam os pulsos de sincronização. Outra alternativa consiste na inclusão dos pulsos de sincronização junto com os bits de informação, utilizando-se alguma forma de codificação. Ambas as técnicas têm sido utilizadas com eficiência.

No entanto, como os caracteres são agrupados em blocos, surge a necessidade de um outro nível de sincronização entre transmissor e receptor: para identificação do início e fim do bloco. Para tanto, usa-se a inserção de um grupo de bits no início do bloco (marca o início da contagem de bits a serem recebidos) e outro no seu final.

Há pastilhas (chips) que podem realizar as tarefas necessárias à formação do bloco de transmissão, inclusão dos caracteres especiais de controle e detecção de erros, denominadas *USART (Universal Synchronous Asynchronous Receiver/Transmitter)*, transmissor/receptor universal síncrono e assíncrono, que também podem realizar as atividades de uma UART.

Entre os interfaces mais modernos desenvolvidos para controlar a transmissão de dados na forma serial existe o que se denomina USB — Universal Serial Bus. Este interface serve para interligar ao processador dispositivos do tipo: joysticks, teclados, scanners, telefones, impressoras, sem que se necessite de interfaces separados para cada um.

O padrão de barramento USB foi desenvolvido por um conjunto de fabricantes de equipamentos de computação: Compaq, DEC, IBM, Intel, Microsoft, NEC e Northern Telecom, tornando-se disponível para o mercado sem ônus, como padrão aberto.

O padrão USB suporta velocidades de até 12 Mbits/segundo, permitindo a conexão de até 127 periféricos, ligando-se a uma única porta de saída no microcomputador.

10.2.2 Transmissão Paralela

Com o uso de transmissão em paralelo, um grupo de bits é transmitido de cada vez, sendo cada um enviado por uma linha separada de transmissão, conforme mostrado na Fig. 10.15. Esse processo já foi descrito no item 6.6, mas é interessante mencionar que sua utilização é mais comum para transmissão interna no sistema de computação (caso dos barramentos analisados no item 6.6.3) e para ligação de alguns periféricos (impressoras, por exemplo) a curta distância, visto que o custo da transmissão paralela é maior em face da quantidade de linhas utilizadas. Quanto maior a distância entre os dispositivos, maior será o comprimento da conexão e maior o custo correspondente. A pastilha Intel 8255A (Programmable Peripheral Interface — PPI) é um bom exemplo de interface usado para controlar a transmissão paralela em microcomputadores.

Um outro interface paralelo muito utilizado na conexão de impressoras é o CENTRONICS, nome dado em razão de o método ter sido desenvolvido por um fabricante com este nome.

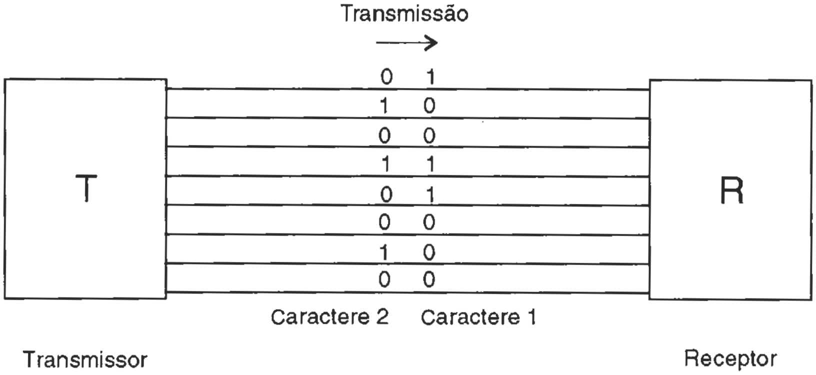


Figura 10.15 Exemplo de transmissão paralela.

O padrão CENTRONICS (padrão porque a maioria dos sistemas utiliza este interface) define um conjunto de sinais que fluem pelas linhas de conexão, bem como estabelece o formato e a quantidade de pontos que devem existir no conector associado. Um outro tipo de interface paralelo denomina-se SCSI — Small Computer Systems Interface), empregado para controlar dispositivos com elevado volume e velocidade de transmissão, como discos magnéticos e CD-ROM, é atualmente utilizado em escala crescente.

10.3 DISPOSITIVOS DE E/S

Um dispositivo de entrada ou de saída (periférico) é o equipamento acoplado a um sistema de computação que efetivamente identifica a função *Entrada* ou a função *Saída*. Um teclado, um monitor de vídeo e uma impressora caracterizam de modo inequívoco aquelas funções, embora sejam apenas parte de um subsistema de E/S. Um teclado, por exemplo, não tem utilidade sem um interface que compatibilize sua lentidão e transmissão bit a bit com a velocidade e transmissão paralela do barramento do sistema.

A seguir vamos efetuar uma breve descrição dos periféricos mais populares, encontrados nos atuais sistemas de computação. No item 10.5, serão descritos outros processos, tecnologia e dispositivos para aqueles interessados em detalhes mais específicos.

Os dispositivos de E/S que descreveremos neste item são:

a) Teclado
b) Monitor de vídeo
c) Impressora
d) Fita magnética
e) Disco magnético
f) Mouse

10.3.1 Teclado

O teclado é um dispositivo de E/S da categoria dos dispositivos que se comunicam com o ser humano, como também o são o vídeo e as impressoras. Nesse caso, eles precisam ser dotados de mecanismos que reconheçam de algum modo os símbolos utilizados pelos humanos (como os caracteres alfabéticos e outros símbolos de nossa linguagem). No caso do teclado, este reconhecimento é realizado pela interpretação do significado elétrico de cada tecla ao ser pressionada. A Fig. 10.16 mostra um exemplo de teclado.

De modo geral, há três categorias de teclado no mercado. Embora todas três funcionem internamente de modo semelhante, elas apresentam aspecto externo diferente (quantidade e tipo de teclas diferentes), e seu uso é particularmente distinto em cada categoria:

1) *Teclados apenas numéricos* — é o caso das calculadoras de bolso ou de mesa, por exemplo (embora seja possível encontrarmos calculadoras de bolso que recebam e processem também caracteres alfabéticos além dos numéricos).

Figura 10.16 Teclado.

2) *Teclados para sistemas dedicados* — consistem em teclas necessárias apenas para a entrada de informações relativas à tarefa do sistema. Por exemplo, o teclado de um sistema de computação para controle ambiental só teria teclas relacionadas ao controle do sistema: ar condicionado, aquecimento, ventoinha, bomba, etc.

3) *Teclado comum para uso geral* — constituído de todas as teclas alfabéticas (permite entrada de caracteres maiúsculos e minúsculos), numéricas, de sinais de pontuação, de operações aritméticas e outras teclas para certas funções especiais. Um teclado desta categoria tem normalmente de 80 a mais de 125 teclas.

Uma tecla pode ser compreendida como uma chave. Ao pressioná-la, estamos acionando a chave e, como conseqüência, algumas ações vão ser realizadas pelos circuitos de controle inseridos no próprio teclado (no caso de teclados de uso geral (categoria 3)). Normalmente, embaixo das teclas há um circuito impresso com vários componentes eletrônicos, inclusive um microprocessador.

Há, atualmente, três tecnologias de fabricação de teclas:

- *teclas mecânicas* (ou de contato direto);
- *teclas capacitivas*; e
- *teclas de efeito-hall.*

Grande parte dos teclados utiliza a tecnologia *capacitiva* e, por essa razão, vamos descrevê-la rapidamente.

A tecla capacitiva funciona na base da variação de capacitância (uma propriedade elétrica) do acoplamento entre duas placas metálicas, variação essa que ocorre quando uma tecla é pressionada.

A grande vantagem desse tipo de tecla é o seu baixo custo e pequeno tamanho, além de não possuir contatos mecânicos, que podem oxidar com o tempo. Tem, portanto, uma vida relativamente longa, cerca de 20 milhões de pressionamentos.

Os teclados funcionam de modo semelhante, embora com variações decorrentes da sua capacidade de teclas, rapidez de resposta desejada e custo. Um teclado típico, o dos microcomputadores atuais, funciona basicamente da seguinte maneira:

a) *Detecção do pressionamento de uma tecla* — um processador interno ao teclado (usualmente os microprocessadores de 8 bits Intel 8048 ou 8049) efetua periodicamente uma varredura para detectar o pressionar de uma tecla.

b) *Realização do "debouncing" do pressionamento* — consiste em confirmar se realmente a tecla foi pressionada. Para tanto, o processador repete várias vezes a varredura sobre a referida tecla.

c) *Geração do código correspondente à identificação da tecla pressionada* — no caso dos microcomputadores PC, isto significa a geração, por um circuito codificador de colunas e linhas, de um código binário (8 bits) referente à tecla pressionada, denominado código de varredura (*scan code*).

d) *Geração de um sinal de interrupção* (ver item 10.4 para explicação sobre o que é uma interrupção) da UCP do microcomputador referente à ação corrente (o pressionar da tecla), de modo a fazer com que a UCP *tome providências relativas à identificação da tecla em questão e seu valor seja passado ao programa* corrente.

e) A UCP troca sinais (relativos à interrupção) com o processador do teclado, para finalmente o código de varredura ser transmitido para uma área de memória principal, onde é interpretado por um programa de E/S residente no microcomputador.

f) O programa em questão (BIOS — Basic Input Output System), sistema básico de entrada/saída, realiza uma detalhada análise no código recebido, para verificar, por exemplo, se a tecla foi pressionada sozinha ou em combinação com outra (como ALT), ou se já existe uma tecla acionada anteriormente (a tecla de letras maiúsculas, por exemplo, CAPS-LOCK) e, finalmente, coloca o código ASCII correspondente na área de memória apropriada, de modo que aquele valor possa ser utilizado pela aplicação com que o usuário esteja trabalhando no momento em que pressionou a tecla.

A grande vantagem dessa metodologia de identificação da tecla pressionada e sua correlação com um determinado código é a possibilidade de se alterar, por programa, o significado da tecla ou conjunto de teclas pressionado. Deste modo, um programa aplicativo, como um processador de textos, pode programar o pressionar das teclas ALT e P para acionar o processo de impressão de um arquivo, enquanto um outro aplicativo pode programar essa mesma atividade (impressão de um arquivo) através do pressionamento de outras teclas.

Os teclados, embora tenham uma funcionalidade que obriga o fabricante a manter as linhas gerais de sua estrutura física (o conjunto de teclas é imutável, embora se possa acrescentar uma ou outra tecla ou alterar seu posicionamento), têm evoluído ao longo do tempo, principalmente em termos de aspecto externo, em face do desenvolvimento de necessidades ergonômicas. Atualmente têm surgido no mercado teclados com formato ergonômico que facilita o apoio das mãos no processo de digitação, principalmente para o caso de pessoas que utilizam o equipamento por longos períodos de tempo. A Fig. 10.17 mostra um desses teclados.

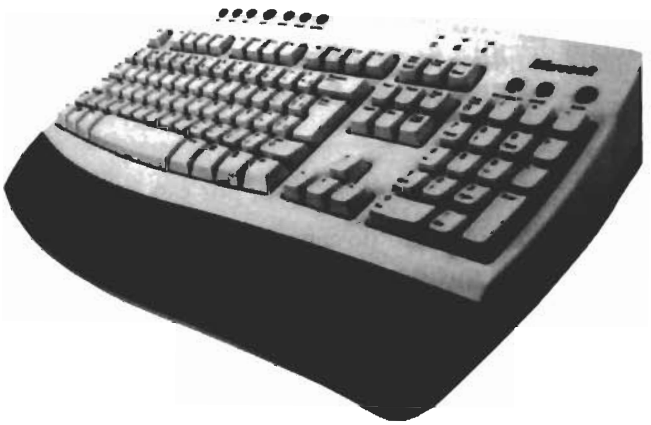


Figura 10.17 Teclado ergonômico.

Basicamente as teclas de caracteres alfabéticos seguem o mesmo padrão, conhecido como QWERTY devido à colocação dessas letras na parte superior à esquerda. O padrão é importante para que os digitadores adquiram prática e velocidade de digitação independentemente do tipo de equipamento utilizado em seu trabalho.

Para mais detalhes sobre teclados, ver o item 10.5.2.

10.3.2 Monitor de Vídeo

Um dos periféricos mais populares (e também um dos mais necessários) é o *vídeo (video display)*. Vivendo na era da informação, necessitamos de ferramentas que nos permitam, da melhor forma possível, encontrar, assimilar e manipular os dados de que precisamos. A maneira mais simples de o ser humano identificar uma informação é através do sentido da visão. Esta é a razão da popularidade e importância vital do monitor de vídeo em um sistema de computação.

Nos primórdios da computação, as informações eram apresentadas muitas vezes na própria forma binária do computador, através de lâmpadas na frente do painel da máquina. Essas lâmpadas, acesas ou apagadas, indicavam o valor 1 ou o valor 0 da informação. Mas, como o ser humano possui uma linguagem para comunicação diferente de 0s e 1s, em pouco tempo apareceram dispositivos para apresentar visualmente a informação com os símbolos mais inteligíveis pelas pessoas.

Na língua inglesa, o termo utilizado para esse componente é *display*, cuja melhor tradução literal seria elemento de exibição, sempre qualificado por outra palavra complementar. Por exemplo, *video display*, que simplificamos para *vídeo* apenas ou *monitor de vídeo* (expressão menos correta, mas também usada).

Há atualmente diversas tecnologias para fabricação de vídeos, que lhes dão diferentes apresentações físicas, sendo, por isso, também usadas em diferentes aplicações. A Fig. 10.18 apresenta um exemplo de tipo de vídeo utilizado em computadores pessoais.

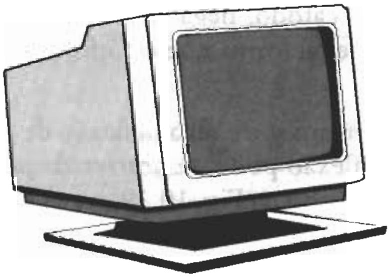


Figura 10.18 Vídeo para computadores.

Os vídeos podem ser classificados quanto à tecnologia de criação e apresentação da imagem, como também quanto à forma com que os bits são passados do sistema para o vídeo. Quanto à tecnologia:

VRC — válvula de raios catódicos (*CRT — cathode-ray tube*);

DEL — diodos emissores de luz (*LED — light emitting diodes*);

VCL — vídeos de cristal líquido (*LCD — liquid-crystal display*);

VPE — vídeos com painel estreito (*TDP — flat panel display*).

Estas são apenas algumas das tecnologias hoje em uso pelos fabricantes de vídeos, sendo que os de painel estreito têm várias modalidades, como de gás plasma e eletroluminescentes.

No entanto, apesar dos avanços atuais, a antiga tecnologia de utilização de VRC prevalece como componente de apresentação visual e, por essa razão, vamos nos deter um pouco na parte mais simples de sua descrição, visto que o princípio eletrônico e os detalhes da fabricação desse dispositivo fogem ao escopo deste livro.

As VRC fazem parte atualmente de milhões de aparelhos de TV, computadores e estações de trabalho em todo o mundo. É a tecnologia padrão para vídeos que outras tecnologias concorrentes vêm tentando suplantar. Por exemplo, a tecnologia de vídeos com painel estreito tem proliferado devido ao crescente número de laptops, notebooks e palmtops que tem surgido no mercado (é comum hoje anúncios de computadores pessoais portáteis — os notebooks — com vídeo de matriz passiva ou de matriz ativa).

O elemento básico de um vídeo do tipo VRC, conforme mostrado na Fig. 10.19, é uma válvula eletrônica (daí o nome de VRC), constituída dos seguintes elementos:

- um catodo, mais comumente denominado canhão de elétrons;
- um anodo, que é a tela frontal, coberta com fósforo;
- um par de bobinas (Yoke), que servem para deflexionar o feixe horizontal e verticalmente.

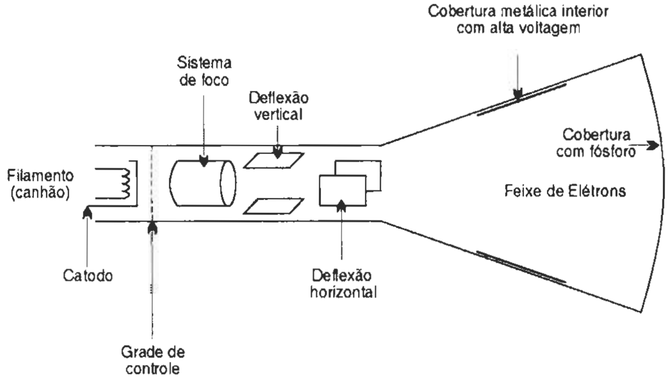


Figura 10.19 Válvula de raios catódicos — VRC.

Seu funcionamento pode ser resumido a seguir:

a) O canhão de elétrons emite os elétrons (um feixe concentrado) que, devido à diferença de potencial elétrico entre ambos os elementos (o catodo, negativo, e o anodo, positivo), caminha em velocidade para a tela frontal, bombardeando-a de tal forma que o fósforo se torna iluminado, aparecendo no local um ponto pequeno e brilhante.

b) No caminho para a tela, o feixe de elétrons sofre uma deflexão de modo a produzir o ponto brilhante em qualquer local que se deseje. Essa deflexão pode ser acarretada por uma corrente elétrica que passe em dois pares de placas, X e Y, como mostrado na Fig. 10.19, ou pode ser realizada magneticamente, através da passagem de corrente elétrica por um par de bobinas orientadas em ângulo reto (horizontal e vertical) e colocadas na parte externa da VRC. Na prática, é mais fácil deflexionar o feixe de elétrons de forma magnética (*Yoke*) do que eletrostaticamente (placas X, Y).

c) Um dos modos de deflexionar o feixe de elétrons (seja com o *yoke* ou por coordenadas X e Y) denomina-se varredura de rastro (*raster-scan*), que, por ser a mais utilizada, será descrita a seguir.

  - Durante a passagem do feixe de elétrons do canhão para a tela, uma voltagem linearmente crescente (em eletrônica denomina-se sinal dente-de-serra, por ser parecido com a forma da lâmina de um serrote — ver Fig. 10.20) é aplicada às placas ou bobinas, acarretando o movimento horizontal, da esquerda para a direita, do feixe de elétrons na tela do vídeo (parte A do sinal na figura). Nas televisões, a intensidade do feixe é modificada (modulada) em cada ponto da tela por onde ele passe, produzindo diferentes intensidade de brilho de acordo com o sinal de recepção captado na antena, formando a imagem recebida da estação de TV que está sintonizada. Nos vídeos dos computadores, o feixe de elétrons tem somente duas opções em cada local: *ligado* (ponto brilhante) ou *desligado*.

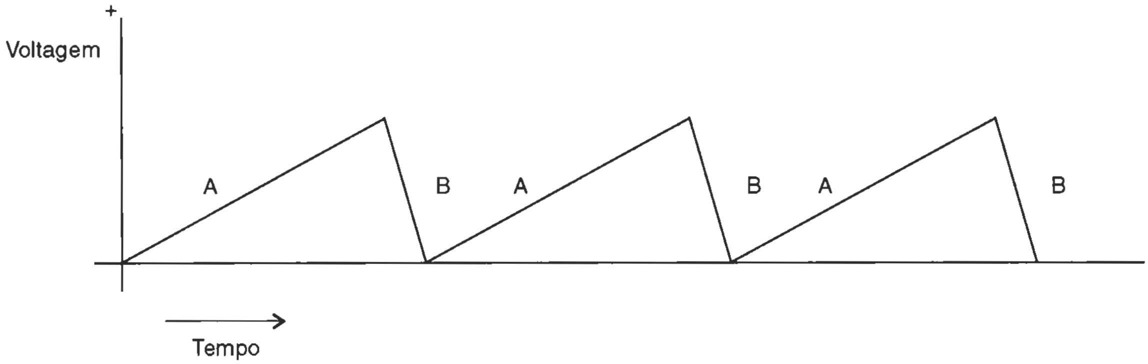


Figura 10.20 Exemplo de sinal dente-de-serra, utilizado para controlar as varreduras horizontal e vertical do feixe de elétrons em uma VRC do tipo *raster*.

- Quando o feixe de elétrons atinge a extremidade direita da tela, ele é desligado e retorna à extremidade oposta (parte B do sinal na figura), para iniciar nova varredura horizontal.
- Esta nova varredura horizontal (parte A seguinte do sinal dente-de-serra) é realizada um espaço para baixo na tela (na prática, dizemos que é uma linha para baixo, já que escrevemos em linhas, tanto na tela quanto no papel), pois o feixe também é deflexionado verticalmente, através de uma voltagem aplicada às placas Y (ou bobinas do *yoke*). A diferença é que, no caso da deflexão vertical, o movimento é mais lento de modo a permitir que o feixe passe por várias linhas, enquanto realiza uma única varredura vertical. Durante um único percurso do sinal dente-de-serra vertical forma-se o que denominamos de moldura vertical, que consiste na contínua deflexão do feixe a cada retorno para iniciar nova linha, até atingir a extremidade inferior da tela, quando a deflexão vertical é desligada para retornar o feixe de elétrons até a extremidade superior esquerda (e reiniciar nova varredura vertical). O conjunto de linhas gerado pelas duas varreduras está mostrado nas Figs. 10.21(a) e (b).

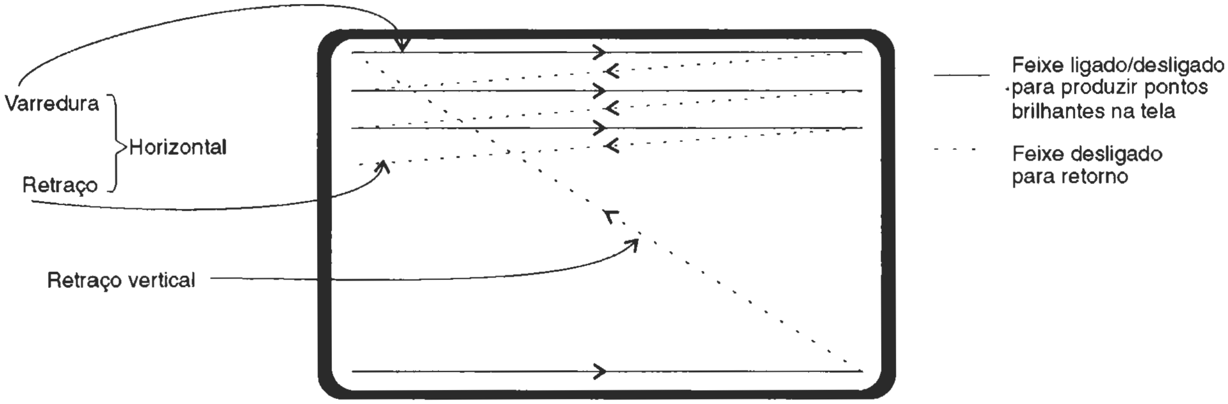

Varredura
Horizontal
Retraço
Retraço vertical
Feixe ligado/desligado para produzir pontos brilhantes na tela
Feixe desligado para retorno

Figura 10.21(a) Funcionamento da VRC. Varreduras horizontal e vertical.

- Fica mais ou menos claro, pela observação da Fig. 10.21, que a freqüência de varredura horizontal é bem maior que a de varredura vertical, isto é, o feixe caminha várias vezes da esquerda para a direita e retorna, enquanto acontece uma única varredura vertical. A quantidade de vezes por segundo que o feixe de elétrons percorre a tela da esquerda para a direita é denominada *freqüência horizontal* (em geral, as TVs possuem este controle externo ao aparelho, podendo ser ajustado pelo usuário). A quantidade de vezes por segundo que a varredura vertical se repete é denominada *freqüência vertical* (da mesma forma, esta freqüência pode ser ajustada dentro de uma determinada margem). A geração dos pontos brilhantes (no caso de vídeo de computador) obtida em uma varredura vertical (que produz várias linhas) é denominada moldura ou *quadro* (*frame*), isto é, uma visão estática no tempo do que foi apresentado na tela durante uma varredura vertical.

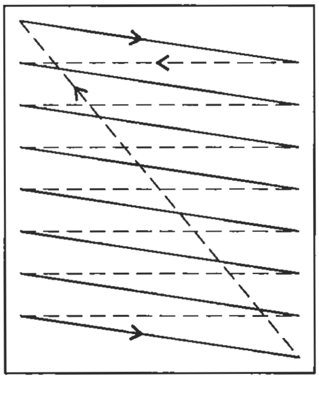


Figura. 10.21(b) Varredura completa (horizontal e vertical) para formar uma moldura (quadro).

- Como era de esperar, não há uma padronização universal para o valor da freqüência horizontal, da freqüência vertical, nem da quantidade de quadros que são formados por segundo. Como a tela não *mantém permanentemente* a luminosidade em um local, decorrente da ligação do feixe de elétrons naquele ponto, é necessário que o feixe passe periodicamente por aquele ponto (e esteja ligado na ocasião). Este procedimento é denominado rescrita da tela (*refreshing*). A rescrita ocorre várias vezes por segundo, de acordo com o valor da freqüência vertical.
- Em geral, um vídeo (para TV ou computador) pode funcionar adequadamente realizando de 50 a 90 varreduras verticais por segundo, ou seja, possuindo uma freqüência vertical de 50 a 90 Hz (a unidade *hertz*, ou ciclos por segundo, já foi discutida no item 6.2.2.2), sendo comum também uma freqüência horizontal entre 15 e 48 KHz. Um conjunto de freqüências, vertical e horizontal, do tipo 50 Hz e 15.625 Hz produz 312,5 linhas em um quadro e 50 quadros por segundo (15.625/50 = 312,5). A Tabela 10.2 mostra um quadro comparativo com alguns sistemas de vídeo e seus valores básicos.

d) É importante mencionar como o feixe de elétrons acende e apaga em seu percurso pela tela, isto é, de onde vem o sinal e como ele se constitui, para controlar o acendimento e apagamento do feixe. Trata-se da informação propriamente dita, armazenada no computador sob a forma binária, mas que pode representar um símbolo que desejamos ver na tela (por exemplo, o caractere "A" ou o símbolo "+", que acabamos de escolher ao pressionarmos as respectivas teclas em um teclado).

Os bits que constituem as informações sobre os símbolos que podemos mostrar na tela de um vídeo estão sempre armazenados em uma memória associada ao sistema de vídeo. Esta memória pode fazer parte do próprio vídeo (como acontece em certos terminais de computadores de grande porte) ou da memória principal do computador (modalidade mais comum nos microcomputadores), ou ainda na memória do interface ou controlador do vídeo (costuma-se usar um termo mais popular — *placa de vídeo*, referindo-se à placa de circuito impresso que contém todos os circuitos eletrônicos que gerenciam a comunicação entre o processador/MP e o monitor de vídeo, gerando os sinais de varredura, enviando as informações de acendimento/apagamento do feixe de elétrons, etc.).

Então, o interface, para apresentar na tela um caractere, armazenado na parte da memória que trata apenas de informações de vídeo, comanda o acendimento do feixe de elétrons nos locais definidos (endereço da tela correspondente ao local desejado para colocar o caractere) para compor o caractere (ver Fig. 10.22). Essa formação do caractere na tela é repetida **V** vezes (rescrita ou *refreshing*), sendo **V** a freqüência horizontal, até que se deseje apresentar outro símbolo, quando, então, o processo se repete.

Há, basicamente, duas modalidades de representação de símbolos em uma tela de vídeo:

- *modalidade textual* (símbolo a símbolo); e
- *modalidade gráfica* (bit a bit).

Tabela 10.2 Quadro Demonstrativo de Valores de Sistemas de Vídeo

Características	Sistema 1	Sistema 2	Sistema 3	Sistema 4
Freqüência horizontal	15.750 Hz	15.625 Hz	31,5 KHz	48 KHz
Freqüência vertical	60 Hz	50 Hz	65 Hz	72 Hz
Número de linhas	262,5	312,5	480	666
Número de campos por segundo	60	50	65	72

10.3.2.1 Modalidade Textual ou Símbolo a Símbolo

Nesta modalidade, a tela de vídeo é dividida em linhas e colunas para formar uma matriz de localização, cada local servindo para armazenar um símbolo válido conforme o código de representação utilizado. A Fig. 10.22 mostra o esquema de uma tela que emprega matriz de 24 linhas por 80 colunas, um dos valores mais comuns em vídeos de computadores, o que permite representar simultaneamente na tela 1920 símbolos.

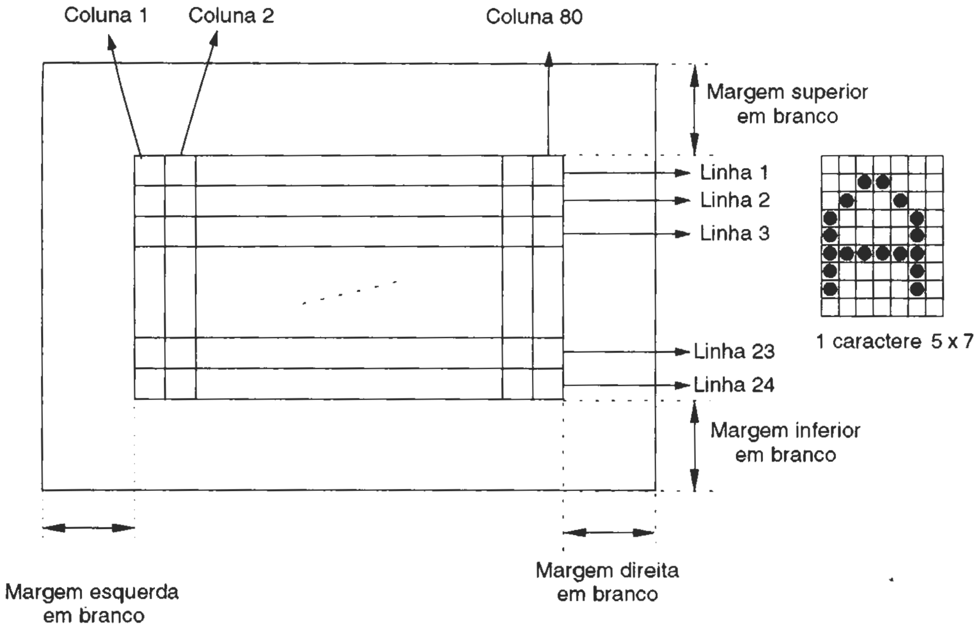
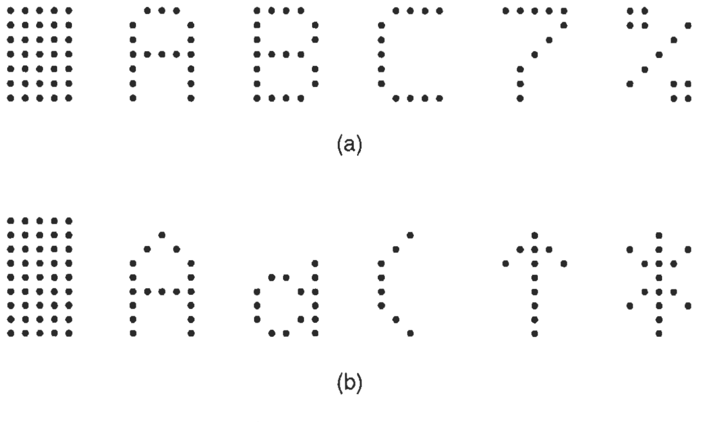

Coluna 1
Coluna 2
Coluna 80
Margem superior em branco
Linha 1
Linha 2
Linha 3
Linha 23
Linha 24
1 caractere 5 x 7
Margem inferior em branco
Margem esquerda em branco
Margem direita em branco

Figura 10.22 Exemplo de uma tabela com vídeo no modo textual, utilizando tecnologia de varredura de rastro.

Cada símbolo é construído por uma matriz de pontos (o ponto brilhante que surge quando o feixe de elétrons acende), em geral sendo do tipo 5 × 7 (5 colunas por 7 linhas), 5 × 9 ou 7 × 9. A segunda tem mais densidade de pontos, servindo para representar caracteres maiúsculos e minúsculos. A Fig. 10.22 mostra um exemplo da apresentação de um caractere por matriz de pontos, enquanto a Fig. 10.23 mostra o exemplo de vários outros caracteres utilizando a matriz 5 × 9.

A matriz de pontos representativa de um caractere é na realidade maior que 5 × 9 ou 5 × 7 ou 7 × 9, visto que há necessidade de serem incluídas linhas e colunas adicionais para separar um caractere do outro, e uma linha da outra. Os monitores monocromáticos (preto e branco) dos IBM PC originalmente utilizavam uma matriz 9 × 14 para representar caracteres, estes em si constituídos da tradicional matriz 7 × 9.


(a)

(b)

Figura 10.23 Exemplos de caracteres formados em um vídeo com matriz de 5 × 7 pontos (a) e 5 × 9 (b).

Para produzir estes pontos, o interface de vídeo deve produzir e emitir as freqüências horizontal e vertical capazes de gerar 14 × 24 = linhas (14 linhas na matriz de 1 caractere vezes 21 linhas de caracteres) mais as linhas de retração (feixe de elétrons apagado, retornando para iniciar nova varredura). Em cada linha, 720 pontos (para permitir 9 pontos por caractere × 80 caracteres).

Cada caractere é armazenado na memória de vídeo (do processador ou do interface) como um conjunto de 2 bytes de informações, um para indicar o código de armazenamento do caractere propriamente dito (código ASCII, por exemplo) e o outro para indicar os atributos de representação do caractere na tela, como, por exemplo, sua intensidade, se estará piscando ou se estará sublinhado.

Desse modo, a memória necessária para armazenar o conjunto de caracteres que podem ser apresentados em uma tela de 25 × 80 é da ordem de 4 Kbytes, isto é, 25 × 80 = 1920 × 2 = 3940 bytes.

10.3.2.2 Modalidade Gráfica ou Bit a Bit

Este tipo de funcionamento modifica o modo com que o interface manipula o acendimento e o apagamento do feixe luminoso durante sua varredura. Acarreta, também, a necessidade de varreduras mais rápidas para melhorar o desempenho do sistema de vídeo.

A tela do monitor é composta de uma única matriz de pontos (diferente da matriz de caracteres, cada um como uma matriz de pontos), que podem estar, em um certo instante, brilhantes (feixe ligado) ou escuros (feixe desligado). Cada ponto ou elemento é denominado *pixel* (abreviação do termo *picture element*). Cada pixel contém 1 bit de informação (indica acender ou apagar) para terminais monocromáticos ou mais bits se o vídeo for do tipo colorido. A Fig. 10.24 mostra uma tela de um monitor de vídeo que funciona no modo gráfico.

O modo gráfico proporciona maior flexibilidade, não só por apresentar caracteres na tela, como também por permitir a formação de qualquer figura através do acendimento de pontos contíguos, na seqüência definida pelo programa aplicativo, pois cada pixel possui um endereço individual. Nesta maneira, podemos apresentar um caractere por uma matriz de 9 × 14 pixels e, em seguida, um outro caractere ao lado, de tamanho menor, por exemplo, por uma matriz de 5 × 7 pixels.

Foi esta flexibilidade que fez surgir as fontes de caracteres (*fonts*), largamente empregadas nos mais diversos aplicativos, como processadores de texto, planilhas eletrônicas, programas que manipulam gráficos e figuras, e outros, já que é possível, em uma mesma tela, utilizar diferentes tamanhos e formas de caracteres, por estes serem construídos com pontos endereçáveis individualmente (pixel a pixel ou bit a bit).

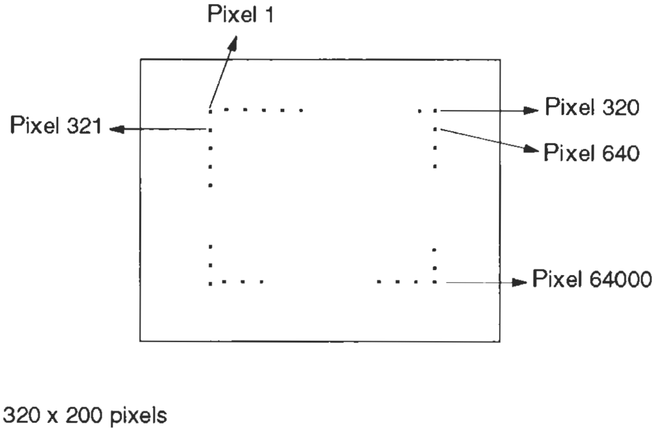


Figura 10.24 Exemplo de um monitor de vídeo funcionando em modalidade gráfica. Cada ponto mostrado "▪" corresponde a um pixel.

Atualmente, a quase totalidade dos sistemas de vídeo funciona quase que exclusivamente no modo gráfico, embora os interfaces existentes permitam o uso de qualquer das duas modalidades (em certos programas pode-se optar por uma ou outra modalidade através do programa). Isto é devido, em grande parte, ao enorme uso do sistema operacional Windows, da Microsoft, que funciona basicamente utilizando um interface gráfico.

Embora o uso do modo gráfico proporcione maior flexibilidade no funcionamento dos aplicativos, ele também acarreta vários problemas de desempenho, demandando hardwares e softwares mais poderosos para solucionar os referidos problemas.

O primeiro deles refere-se à capacidade de memória necessária para armazenar as informações a serem apresentadas em uma tela (em geral, devem-se apresentar várias telas para proporcionar mais rapidez ao funcionamento de um aplicativo). Foi mostrado anteriormente que, na modalidade textual, necessita-se de cerca de 4K de memória para apresentar uma tela no vídeo.

No caso do modo gráfico, para representar uma matriz de 320 × 200 pontos, há necessidade de serem armazenados 64.000 pontos, cerca de 8 Kbytes e 100K para representar uma única tela com resolução de 1024 × 768 pontos em preto-e-branco.

Na prática, os requisitos de memória de vídeo são bem maiores, pois é preciso armazenar várias telas e não apenas uma (senão o sistema teria um baixíssimo rendimento), e além disso o emprego de vídeo colorido é quase uma unanimidade, o que requer 1 ou até 3 bytes de informação para cada pixel.

O outro problema está relacionado com o desempenho do sistema de vídeo, que precisa ser bem maior do que o necessário a um sistema que funcione na modalidade textual, devido justamente à quantidade de bits envolvida e ao modo de funcionamento do interface.

No modo gráfico, cada informação é movimentada da memória de vídeo para a tela em grandes quantidades de bits. Por exemplo, uma tela do programa Windows pode conter um retângulo, desenhado com o uso de uma matriz de 30 × 12 pontos (ou pixels) = 360 pontos (360 bits ou 45 bytes). Se existisse somente este retângulo e quiséssemos movê-lo para outro ponto da tela, todos os 45 bytes teriam que ser enviados, além do cálculo de endereços, que também é uma tarefa trabalhosa.

Se, por outro lado, estivéssemos utilizando um processador de texto, e toda a tela estivesse preenchida com caracteres do texto e quiséssemos "rolar" uma linha do texto (desaparecer a linha superior e aparecer uma linha nova no fundo da tela), seria necessário recolocar todos os bits que formam todos os caracteres de todo o texto mostrado (reescrever toda a tela sem a linha superior e com a inclusão da nova linha de baixo). Isto requer elevada velocidade de processamento, entre outros atributos. Razão por que atualmente os interfaces de vídeo ou placas gráficas possuem grande capacidade de memória (1 ou 2 Mbytes) e até processadores completos, para reduzir a sobrecarga da UCP e acelerar o processamento das telas. Por isso mesmo elas são chamadas *placas aceleradas de vídeo.*

10.3.2.3 Vídeo Colorido

Há uma diferença considerável de tecnologia e requisitos de desempenho de sistemas de vídeo entre aqueles que processam e apresentam informações de forma monocromática (com o emprego de apenas uma cor) e os que mostram as informações em cores. No entanto, neste tópico, abordaremos de forma simples o assunto.

Em primeiro lugar, como a imagem é construída. Sabemos que os sistemas monocromáticos funcionam com um canhão que produz um feixe de elétrons, que acende e apaga um ponto luminoso na tela (no modo gráfico é um pixel). No caso de vídeos coloridos, há necessidade de três canhões, que geram um feixe de elétrons para cada cor fundamental — vermelho, verde e azul. Os feixes são acionados para ligarem e desligarem de modo a produzir diferentes cores.

Para desenhar uma imagem na tela, os feixes caminham da extremidade superior esquerda para a extremidade inferior direita, repetindo-se esta varredura cerca de 40 a 90 vezes por segundo (freqüência vertical), dependendo do grau de persistência do fósforo na tela e do tipo de varredura realizada (ver item 10.5.3 para maiores detalhes sobre a formação dos pixels). Cada pixel é constituído de uma tríade de pontos vermelhos, verdes e azuis, sendo também utilizado como unidade de referência de resolução de um vídeo/interface. Variando-se a intensidade de cada cor, pode ser produzida qualquer cor que se queira, e se os três feixes forem iluminados com igual intensidade, obtém-se uma cor resultante branca, caso a observação tenha sido realizada a uma distância razoável.

10.3.2.4 Algumas Observações Finais

Modo Entrelaçado e Não-entrelaçado (*Interlaced* e *Non-interlaced*)

Quando descrevemos o funcionamento de um vídeo com varredura de rastro, mencionamos que o feixe de elétrons (esta explicação é válida seja um feixe, no caso de vídeos monocromáticos, ou três feixes, para o caso de vídeos coloridos ou policromáticos), ao realizar a dupla varredura, vertical e horizontal, produz um *quadro* (*frame*) constituído de várias linhas. Um conjunto de quadros gerados por segundo produz uma determinada imagem ou mantém a que está sendo apresentada. Há duas maneiras de obter-se a referida imagem:

Modo entrelaçado — é a técnica pela qual o feixe eletrônico varre a tela produzindo a metade das linhas que constituem um quadro. Por exemplo, se um quadro é constituído de 530 linhas, na primeira varredura são apenas geradas as linhas ímpares (1, 3, 5 ...) e em uma segunda varredura o quadro se completa com a geração das linhas pares. A vantagem desta técnica reside na redução de custos dos circuitos eletrônicos e do sistema como um todo em virtude da necessidade de freqüências mais baixas de varredura. No entanto, devido à capacidade de permanência de luminosidade do fósforo que recobre a tela, ocorre uma cintilação da imagem (*flickering*), que varia de intensidade de acordo com o tempo de persistência dos pontos das linhas ímpares sobre os pontos das linhas pares. Isto é uma desvantagem caso se desejem imagens nítidas, sendo a cintilação tão mais acentuada quanto mais pixels são gerados na tela (resolução maior).

Modo não-entrelaçado — é a técnica utilizada para evitar a cintilação e produzir imagens mais perfeitas, especialmente quando se empregam sistemas de vídeo de alta resolução. Por esta técnica, cada quadro é montado em apenas uma passagem, isto é, em apenas uma varredura vertical, de modo que deixa de ocorrer diferença de persistência. Isto é obtido com freqüências mais elevadas de varredura, entre outras características. O problema desta técnica, claramente superior à técnica entrelaçada de geração de quadros quanto a desempenho visual reside no seu custo. Para funcionarem com a técnica de não-entrelaçamento, placas de vídeo e monitores são bem mais caros, devido em grande parte às altas freqüências envolvidas.

Resolução

De modo geral, a resolução de um sistema de vídeo é medida pela quantidade de pixels que pode ser apresentada em uma tela. Esta quantidade é descrita em termos de dois valores: a quantidade de pixels mostrados horizontalmente (em uma linha) e a quantidade de pixels mostrados verticalmente (em uma coluna). Por exemplo, uma resolução de 320 × 200 (o número à esquerda é usualmente maior) significa que na tela são endereçáveis 320 pixels em cada linha por 200 pixels em cada coluna, um total de 64.000 pixels. Além desse valor, uma outra medida *dot pitch* (afastamento entre pontos) é também considerada para compor medidas de resolução de um vídeo.

A resolução de um sistema de vídeo em microcomputadores está incluída dentro de um conjunto de definições (freqüência horizontal, freqüência vertical, largura de faixa, etc.) que constitui um determinado padrão. A indústria de microcomputadores popularizou estes padrões e, assim, é comum ouvirmos as pessoas falando em adquirir um computador com vídeo VGA ou SVGA, sem entender o que isto realmente significa.

Embora não faça parte de qualquer padrão, a medida de *dot pitch* sempre deve ser considerada quando se examinam características de um monitor de vídeo. O *dot pitch* é a distância existente entre dois pixels adjacentes ou a distância entre dois pontos coloridos de uma tríade. É medida em milímetros (mm) e atualmente bons vídeos se apresentam com valor de *dot pitch* entre 0,24 mm e 0,40 mm.

Vídeos oferecidos com baixo valor de *dot pitch* (um valor típico é 0,28 mm) e alta resolução (grande quantidade de pixels, e um bom valor atualmente é 800 × 600) produzem imagens mais nítidas e definidas.

Os padrões de resolução atualmente existentes no mercado de vídeos para microcomputadores são:

VGA — Video Graphics Array — surgiu em 1987 com o lançamento dos sistemas PS/2 da IBM. Compreende vários padrões de medidas, dos quais o mais utilizado é 640 × 480 pixels, com 16 cores (significa que pode apresentar na tela 16 cores diferentes), freqüência vertical de 70 Hz e freqüência horizontal de 31,5 kHz.

SVGA — Super VGA — surgiu em 1989 e compreende uma medida de 800 × 600 pixels, com 16 cores e uma faixa de freqüências verticais entre 56 e 72 Hz, com freqüências horizontais entre 31,5 e 48,0 kHz.

8514A — surgiu também em 1987, definida pela IBM para seus sistemas de vídeo, e aceita resoluções de 640 × 480, 1024 × 768 e até 1280 × 1024 pixels, com 16 e 256 cores.

Há outros valores de resolução mais elevados (e sem um nome específico) e que já vêm sendo empregados em escala crescente, como os de 1024 × 768 pixels e 1280 × 1024 pixels.

Há também outros padrões, alguns já em desuso, como CGA, EGA, MGA e outros, XGA, não muito populares por serem proprietários. Outros detalhes sobre sistemas de vídeo podem ser encontrados no item 10.5.3.

10.3.3 Impressoras

Assim como o monitor de vídeo, a impressora é o periférico clássico de saída, onde as informações armazenadas internamente no computador sob a forma binária são de algum modo convertidas em símbolos impressos em um meio externo qualquer (o papel é o mais comum) e em um formato inteligível ao usuário.

Ao se examinar uma impressora, devem ser consideradas algumas características básicas que podem definir seu desempenho em relação a outros dispositivos. Algumas dessas características são:

- O volume de impressão que ela suporta em uma unidade de tempo. Impressoras podem indicar sua vazão de impressão em caracteres por segundo (cps), em linhas por minuto (lpm) e em páginas por minuto (ppm), dependendo da tecnologia que utilizam para enviar os símbolos para o meio de impressão.
- A tecnologia utilizada para gerar os símbolos a serem impressos. Atualmente, impressoras podem ser do tipo:
  - de impacto (de esfera, de margarida e matricial);
  - sem impacto;
  - de jato de tinta;
  - a laser;
  - por transferência de cera aquecida (*thermal-wax*);
  - por sublimação de tinta (*dye sublimation*).

10.3.3.1 Impressoras Matriciais

Entre as impressoras de impacto, descreveremos apenas as *matriciais*, visto que as impressoras com mecanismo de impressão de esfera e de margarida são fabricadas para emprego em máquinas de escrever.

As impressoras matriciais, muito populares no mercado de microcomputadores, fazem parte da categoria de máquinas denominadas impressoras de impacto. Seu mecanismo de impressão consiste em um dispositivo qualquer (depende do tipo, mas pode ser, por exemplo, um conjunto de *martelos ou agulhas*, usado pelas impressoras matriciais) que se projeta contra uma fita com tinta, imprimindo o símbolo no papel que está atrás, conforme mostrado na Fig. 10.25.

O nome da tecnologia de impressão já caracteriza o seu funcionamento básico, isto é, os caracteres são formados por uma matriz de pontos (daí o nome matricial — em inglês usa-se *dot pitch*), de modo semelhante à matriz descrita no item anterior.

A Fig. 10.25 mostra um esquema do funcionamento de uma impressora matricial. Nela aparece: o mecanismo de impressão, constituído pela cabeça de impressão (A), a fita impregnada com tinta (B) e o papel em que os símbolos serão impressos (C).

O método de geração dos pontos no papel se inicia com a existência de um dispositivo (cabeça de impressão) composto de vários fios, muito finos, as *agulhas ou pinos* (em inglês usa-se o termo *pin*), montados em um tubo e ligados a uma bobina eletromagnética. As agulhas, que podem variar, em quantidade, entre 9 e 24, são dispostas verticalmente, formando uma coluna, quando se trata de cabeça de impressão com 9 agulhas, ou 3 colunas de 8 agulhas cada, quando se trata de cabeça de impressão de 24 agulhas. Para que as agulhas possam ficar dispostas bem próximas umas das outras (e garantir, assim, boa qualidade de impressão — boa resolução), os magnetos são usualmente arranjados de forma radial, como mostrado nas Figs. 10.25 (parte indicada por A) e 10.26, visto que a largura do tubo é maior que a das agulhas. Cada agulha tem um diâmetro da ordem de 0,2 a 0,3 mm.

A *cabeça de impressão* caminha da esquerda para a direita (ou nos dois sentidos, dependendo do tipo de impressora) e em seu percurso vai marcando os pontos correspondentes aos caracteres que se deseja imprimir.

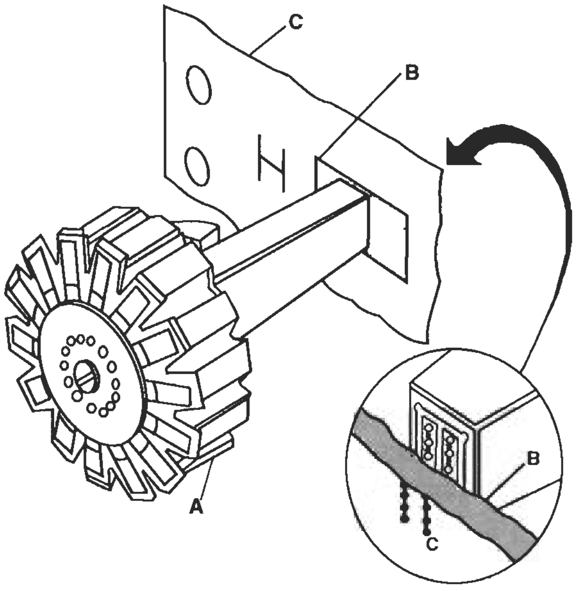


Figura 10.25 Mecanismo de impressão de uma impressora matricial.

Em geral, um caractere é constituído de uma matriz com 5 × 9 pontos (impressora com 9 agulhas) ou bem mais, no caso de impressoras de 24 agulhas. Quando um padrão de bits, correspondente a um caractere, é recebido no circuito de controle da impressora, este padrão gera correntes elétricas que vão acionar a bobina ligada à correspondente agulha. Nessa ocasião, a bobina energizada projeta rapidamente a agulha, que impacta a fita com tinta, impregnando o papel com um ponto. Logo em seguida, uma mola retorna rapidamente a agulha, que fica pronta para novo acionamento. Dessa forma, a cabeça imprime simultaneamente os *n* pontos de uma coluna e logo em seguida os *n* pontos da coluna seguinte, e assim sucessivamente, até formar todo o caractere e o caractere seguinte e o seguinte, até completar a linha.

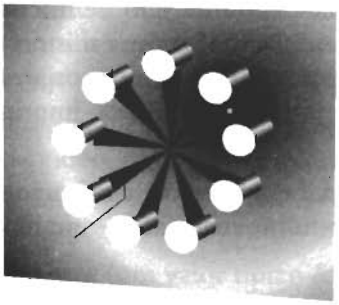


Figura 10.26 Distribuição radial das agulhas em uma cabeça de impressão.

Tendo em vista que o equipamento imprime um caractere de cada vez, costuma-se medir a velocidade de impressão em cps — caracteres por segundo. Embora seja cada vez menor o lançamento de impressoras matriciais, em face do baixo custo e da boa resolução das impressoras de jato de tinta, principalmente, ainda podemos mencionar que valores típicos de mercado variam entre 50 cps até impressoras com cerca de 460 cps. Impressoras matriciais também podem imprimir em cores, utilizando-se para isso fitas de impressão com cores diferentes.

Apesar de ainda produzidas em escala razoável, as impressoras matriciais vêm perdendo usuários em face das vantagens de preço/desempenho de modelos com tecnologias mais avançadas, especialmente as impressoras de jato de tinta.

10.3.3.2 Impressoras de Jato de Tinta

As impressoras de jato de tinta (*ink-jet*) produzem caracteres em um papel em forma de matriz de pontos, como nas impressoras matriciais. A diferença está na técnica de criar o ponto no papel. No caso de impressoras de jato de tinta, o ponto é o resultado de uma gota de tinta que é depositada no papel e secada por calor para não escorrer. Muitas dessas gotas depositadas moldam o formato do caractere, de modo semelhante aos pontos obtidos pela projeção das agulhas em impressoras matriciais.

O mecanismo de impressão é, em geral, constituído de uma certa quantidade de pequeninos tubos com um *bico* apropriado para permitir a saída das gotas de tinta (em inglês denomina-se *nozzle*). Um valor típico de bicos existentes em mecanismos de impressão dessas impressoras oscila entre 50 e 64, mas atualmente já estão sendo lançados novos modelos com 128 e até 256 bicos. A tecnologia mais comum — *drop-on-demand buble jet* — projeção gota a gota por demanda — consiste na passagem de uma corrente elétrica por uma resistência, que, aquecida por esta corrente, gera suficiente calor para o tubo de tinta. No instante em que se aquece o suficiente, a tinta vaporiza e se expande, acarretando a saída de uma gota pelo bico do tubo, a qual vai ser depositada e secada no papel, gerando um ponto de tinta. O processo ocorre milhares de vezes por segundo durante a impressão.

Há impressoras que funcionam com apenas um tubo de tinta preta — impressoras do tipo monocromático — e outras, cuja quantidade de modelos vem crescendo continuamente, que imprimem colorido através do emprego de 3 ou 4 tubos de tinta (com 3, há um tubo de tinta magenta, um tubo de tinta cyan (azul) e outro de tinta amarela; com 4 tubos, acrescenta-se um tubo de tinta preta).

Sendo uma impressora do tipo matricial, sua resolução (a quantidade de pontos que constituem um caractere) é tão maior — produz caracteres mais sólidos e nítidos — quanto a quantidade de bicos que o mecanismo de impressão pode ter. Seu mecanismo de impressão contém algo em torno de 60 bicos, produzindo, assim, uma matriz de pontos muito mais densa do que se consegue com impressoras matriciais de 24 agulhas. Valores típicos de resolução de impressoras de jato de tinta estão na faixa de 300 × 300 pontos por polegada e 360 × 360 pontos por polegada (*dpi — dots per inchs*), com caracteres constituídos de uma matriz de 18 × 48 e até 36 × 48 pontos. Elas possuem outra vantagem sobre as impressoras matriciais: são silenciosas, já que não dispõem de mecanismo de impacto.

10.3.3.3 Impressoras a Laser

As impressoras que funcionam utilizando-se de um mecanismo de impressão a laser têm se tornado muito populares com o advento dos modelos pessoais, para microcomputadores. O desenvolvimento da tecnologia de impressão a laser, impulsionado por uma crescente demanda de mercado, tem tido como resultado a fabricação de impressoras deste tipo com um custo/desempenho extraordinário. Atualmente podem ser encontrados no mercado impressoras a laser capazes de imprimir 4 páginas por minuto (4 ppm), 8 ppm, 12 e mais, com excelente qualidade e preço inicial em torno de R$350,00.

O mecanismo de impressão funciona de modo semelhante ao das copiadoras de documentos. Isto é, a idéia consiste em formar, em um cilindro fotossensitivo, uma imagem da página que será impressa. Em seguida, um produto chamado "toner", composto de partículas minúsculas, é espalhado sobre a imagem criada no cilindro. Finalmente, a imagem é transferida do cilindro para um papel e secada por intenso calor; depois disso, o cilindro deve ter a imagem apagada para que uma nova imagem possa ser nele criada. E assim, sucessivamente, as páginas vão sendo impressas. A imagem é criada no cilindro através de um feixe de laser que é aceso e apagado a cada ponto do cilindro (como pixels em um vídeo), conforme a configuração binária e a localização dos símbolos que se deseja imprimir.

A Fig. 10.27 mostra um esquema simplificado do funcionamento de um mecanismo de impressão a laser. Na Fig. 10.27(a), podemos observar o cilindro fotossensitivo, que é envolvido por um dispositivo que gera uma carga elétrica negativa no cilindro (*drum corona wire*). Quando o cilindro é exposto à luz (feixe de laser ou outro meio), a carga elétrica em sua superfície se altera, criando uma área de exposição na qual a imagem é produzida pelo feixe de laser ou qualquer outro meio semelhante. O tambor vai girando e, assim, o feixe luminoso vai criando os pontos, que formam os símbolos, linha por linha.

Ao mesmo tempo, um cartucho, cheio de pequenas partículas de pó preto, fabricado com material sensível eletricamente, vai depositando estas partículas no *cilindro*, nos pontos que tiveram sua carga elétrica alterada (e que irão compor os símbolos). Em seguida, o papel que está passando pelo cilindro recebe uma carga elétrica suficiente para atrair as partículas do toner nos pontos desejados, formando, assim, os símbolos. Logo após, o papel passa por dois dispositivos aquecedores (funcionam com elevada temperatura) para secar o toner no papel.

A Fig. 10.27(b) mostra o esquema de uma das tecnologias utilizadas para produzir os pontos no cilindro, a tecnologia de laser. Nesta tecnologia, um diodo gera um feixe que é refletido em um tipo de espelho (*spinning mirror*); o feixe varre o cilindro e é rapidamente aceso e apagado, conforme os bits de informação que chegam ao mecanismo, para compor os pontos (feixe ligado) que constituirão a imagem que será impressa.

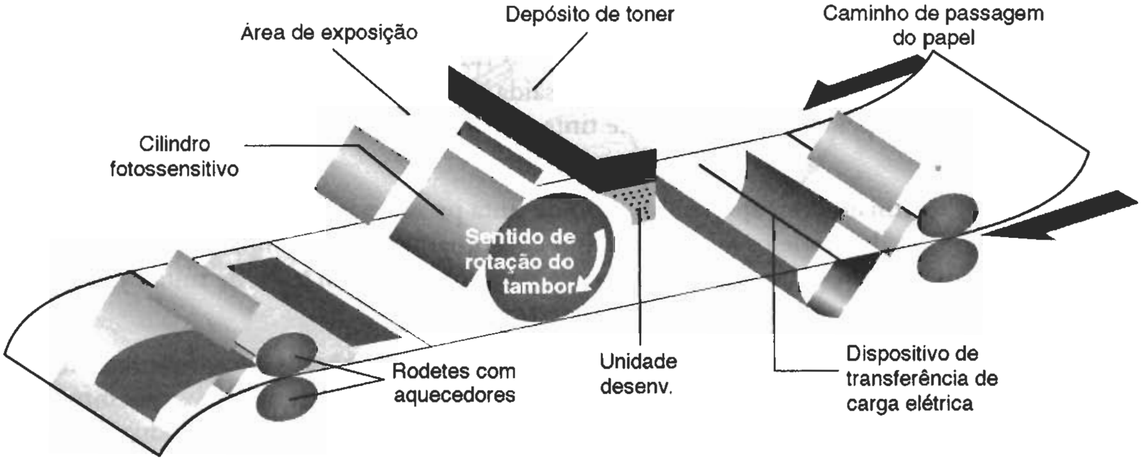

Área de exposição
Depósito de toner
Caminho de passagem do papel
Cilindro fotossensitivo
Sentido de rotação do tambor
Rodetes com aquecedores
Unidade desenv.
Dispositivo de transferência de carga elétrica

Figura 10.27(a) Mecanismo de impressão a laser.

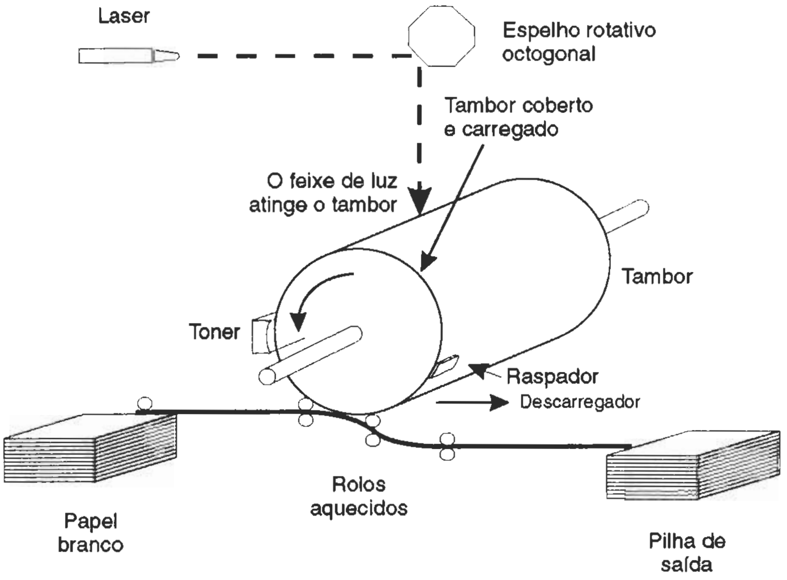

Laser
Espelho rotativo octogonal
Tambor coberto e carregado
O feixe de luz atinge o tambor
Tambor
Toner
Raspador
Descarregador
Rolos aquecidos
Papel branco
Pilha de saída

Figura 10.27(b) Esquema de impressão com feixe de laser.

As impressoras a laser também imprimem ponto por ponto e, por essa razão, sua resolução é medida em *pontos por polegada* (*dpi — dots per inch*). Há, no mercado atual, impressoras deste tipo funcionando com resolução de 300 dpi, 600 dpi e 1200 dpi, produzindo páginas em uma taxa em torno de 4 ppm e 8 ppm (impressoras pessoais), como também 20 e mais (impressoras que funcionam em redes locais de microcomputadores) ou máquinas de maior porte, capazes de imprimir mais de 80 ppm.

10.3.4 Fitas Magnéticas

A fita magnética é um dos meios mais antigos de armazenamento de informações em computador, servindo também como meio de entrada e saída de programas e dados. O dispositivo eletromecânico, capaz de ler e gravar as informações nas fitas, é o periférico (unidade de fita magnética ou driver), enquanto a fita em si é o meio de armazenamento.

O princípio de funcionamento das unidades de fita magnética de computadores é bastante semelhante ao dos tape-deck de som, consistindo em dois carretéis (a fita com dados sempre se desenrola de um lado para o outro, da esquerda para a direita, onde está o carretel alimentador) e a fita que passa por um par de cabeças de leitura e gravação em velocidade constante. A Fig. 10.28 mostra um mecanismo de acionamento das fitas magnéticas usadas em computadores de grande porte até a década de 1980.

Devido ao conceito de funcionamento, as fitas são dispositivos de acesso apenas seqüencial, isto é, cada informação é armazenada após a última. Sua recuperação (leitura) é realizada através de um processo também seqüencial, a localização do registro desejado começa a partir do início da fita; de registro em registro, até que seja identificado o registro desejado (como usualmente também fazemos com o gravador cassete de som. Nesse caso, quando queremos ouvir uma música que se encontra gravada no meio da fita, começamos do princípio, desenrolando a fita até o ponto desejado).

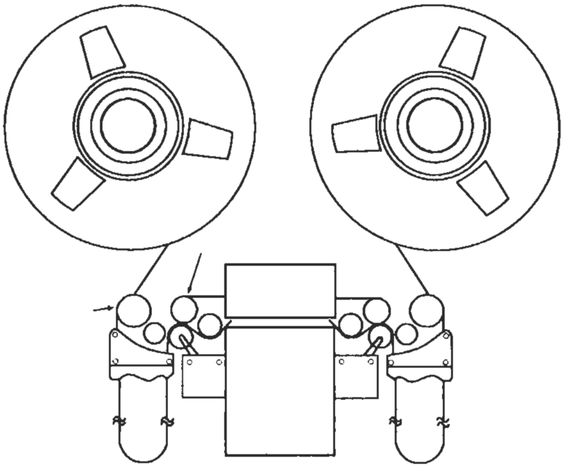
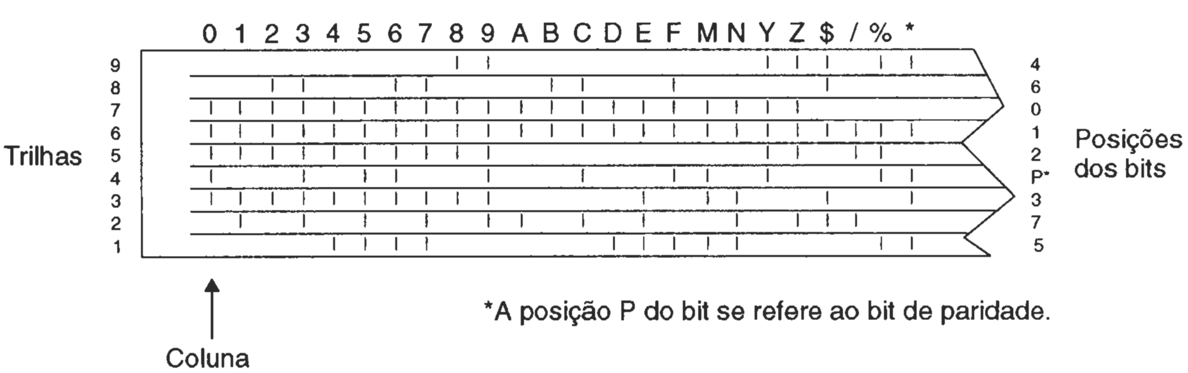


Figura 10.28 Mecanismo de transporte de uma unidade de fita magnética.

Uma fita magnética é normalmente constituída de uma tira contínua de material plástico coberto com elementos magnéticos, onde os dados (bits) são gravados como campos magnéticos (em um sentido representam o bit 0, no outro sentido representam o bit 1). Esses campos são gerados pela passagem de corrente elétrica em uma bobina existente na cabeça de gravação. A Fig. 10.29 mostra um trecho de fita magnética retificada, onde se observam os bits representados em linhas paralelas (canais).



*A posição P do bit se refere ao bit de paridade.

Figura 10.29 Parte retificada de uma fita magnética, mostrando os bits, de cada caractere armazenados em colunas (coluna de 9), cada bit em uma linha diferente (trilha).

Em geral, as fitas magnéticas para computadores são enroladas em carretéis com comprimento de 300 pés, 600 pés, 1200 pés e 2400 pés.

Conforme mostrado na Fig. 10.29, os dados são armazenados em canais paralelos, denominados trilhas, que percorrem toda a fita. O número de trilhas pode ser igual a 7 ou 9, embora fitas e acionadores com 7 trilhas tenham ficado obsoletos há décadas; somente fitas com 9 trilhas continuaram a ser fabricadas.

A razão da escolha desses números advém do padrão de bits de cada código de caracteres usado no mercado (antigamente usava-se o código BCD de 6 bits por caractere; em seguida, os códigos mais populares passaram a ser o ASCII e o EBCDIC, que representam caracteres com 8 bits por caractere — 1 byte, mesmo o ASCII de 7 bits, que acrescenta um bit adicional, sem efeito, para manter o total múltiplo de 2 (ver Cap. 7)).

Cada caractere é armazenado verticalmente, um bit por trilha, mais um bit de verificação (bit de paridade), completando as 9 trilhas (o conjunto de bits de uma coluna é também conhecido como quadro ou *frame*).

Uma das características mais interessantes do sistema de transporte das unidades de fita magnética consiste na rapidez de parada e partida da rotação dos carretéis. É importante que o carretel inicie a rotação e pare rapidamente, visto que o processo de leitura e gravação somente se inicia quando a velocidade de passagem da fita pelo cabeçote for constante (senão, poderá haver erro de leitura/gravação); quanto mais rápida for a partida, mais rápido a velocidade de rotação se estabiliza e não há atrasos acentuados na operação de leitura ou de gravação.

O espaçamento entre as colunas (que é o mesmo espaço entre bits por trilha e entre caracteres) é obtido automaticamente durante a operação de gravação e varia de acordo com a velocidade de passagem da fita, indicando uma das principais características do desempenho de unidades de fita — *sua densidade.*

É possível utilizar densidades de 800 caracteres por polegada (usa-se o termo bpi — *bytes per inch*, visto que um caractere é codificado por byte), 1600 bpi e até 6250 caracteres (ou bytes) por polegada. Essa última, devido à grande capacidade de armazenamento, é a que mais vem sendo empregada.

Em meados da década de 1980, foi desenvolvido um novo tipo de fita denominado cartucho, cuja característica principal é a enorme densidade de gravação, mais de 30.000 bpi, o que lhe garante grande capacidade de armazenamento. É um sistema apropriado para servir de back-up para unidades maiores de disco magnético.

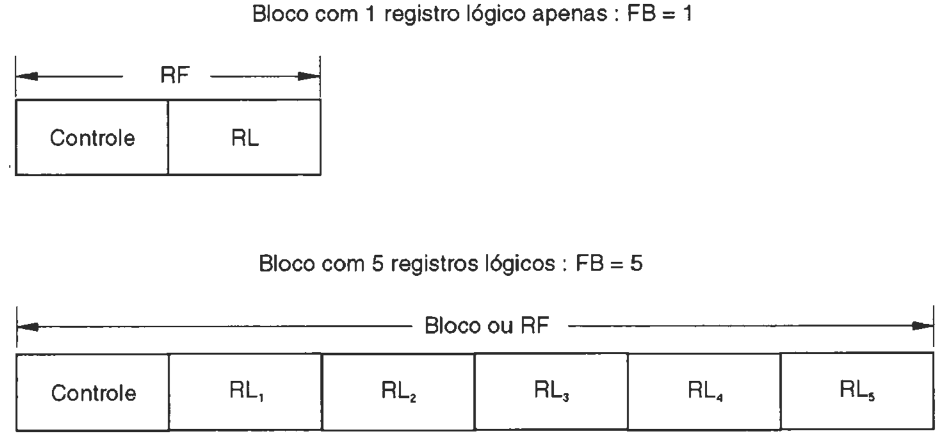
Em face da baixa velocidade de acesso em relação aos discos, as fitas magnéticas deixaram de ser elementos de E/S para processamento, sendo mais empregadas como meio de armazenamento off-line, via de regra servindo para obter-se cópias de segurança de dados armazenados em discos (back-up dos arquivos armazenados nos discos magnéticos).

As fitas magnéticas costumam ter uma marca refletora no seu início e no final do carretel (uma pequena peça metálica que é detectada por um sensor ótico) de modo a impedir que a fita ultrapasse os limites e se estrague.

Basicamente, as informações (dados ou programas) são armazenadas em blocos ou registros físicos (conjunto de registros lógicos — ver Cap. 2), separados por espaços denominados IRG (Inter Record Gap), ou simplesmente *gap*. Os gaps são incluídos entre blocos para permitir a aceleração e desaceleração da fita sem haver perda de leitura do início de cada novo bloco. Como há um gap entre cada par de blocos armazenados, quanto maior a quantidade de blocos, maior será a quantidade de gaps e, conseqüentemente, de espaço morto na fita. Isto porque, embora indispensável (devido ao já mencionado problema de desaceleração/aceleração), o gap é um espaço perdido, que reduz a disponibilidade da fita para armazenar informações úteis.

Bloco ou registro físico (RF) é a unidade de armazenamento/transferência de informação dos sistemas de fita magnética, sendo constituído de um ou mais registros lógicos (RL) e mais alguns bytes inseridos pelo programa de controle (parte do sistema operacional) com dados necessários à identificação e recuperação do bloco (ver Fig. 10.30). A quantidade de RL em um bloco (RF) define um elemento chamado fator de bloco — FB.

Vamos imaginar um exemplo de armazenamento/recuperação de arquivo em fita magnética, de modo a verificar o efeito do tamanho do bloco (decorrente da escolha de um valor de fator de bloco) no desempenho de um sistema de fita magnética.


Bloco com 1 registro lógico apenas : FB = 1
RF
Controle
RL
Bloco com 5 registros lógicos : FB = 5
Bloco ou RF
Controle
RL_1
RL_2
RL_3
RL_4
RL_5

$$\text{Fator de Bloco} = \text{FB} = \frac{\text{N° de RL}}{\text{N° de RF}} = \frac{\text{NRL}}{\text{NRF}}$$

Figura 10.30 Blocos (RF) e registros lógicos (RL).

Suponhamos que um certo arquivo constituído de 10.000 RL esteja armazenado em uma fita magnética e que o referido arquivo tenha sido armazenado usando-se um FB igual a 5. Como:

FB = NRL/NRF e sendo NRL = 10.000 e FB = 5

Então:

NRF (n.° de blocos) = NRL/FB = 10.000/5 = 2000 blocos.

O sistema armazenará 2000 blocos e gastará espaço com cerca de 2000 gaps (na realidade seriam 1999 gaps entre o primeiro e o último bloco). Consumirá na leitura/gravação de todo o arquivo um tempo correspondente à operação de transferência de 2000 blocos.

Caso fosse modificado o fator de bloco para um valor igual a 10, teríamos um novo cálculo para o armazenamento do mesmo arquivo:

NRF = NRL/FB = 10.000/10 = 1000 blocos.

O sistema armazenará o arquivo utilizando a metade da quantidade de blocos do caso anterior, pois agora cada bloco possui 10 RL em vez de 5 RL. Com isso, a quantidade de gaps também se reduz pela metade (1000 gaps), com grande economia de espaço morto.

Conforme se pode verificar dos exemplos anteriores (os cálculos foram simplificados para indicar apenas os dados necessários à explicação), o aumento do valor do FB (aumento do tamanho dos blocos) acarreta uma dupla vantagem:

- *menor tempo de transferência de dados entre fita e MP* — a maior parte do tempo de transferência reside nos procedimentos de localização do bloco e movimento do carretel para posicionamento da fita sobre bloco desejado e não no tamanho do bloco;
- *maior economia de espaço perdido com gaps* — sobrando, assim, mais espaço para armazenamento de informações úteis.

Com os elementos utilizados e sabendo-se que, em geral, um gap ocupa 3/4 de polegada (0,75") de uma fita, podemos calcular (valores aproximados apenas) o consumo de espaço ocupado por um arquivo em fita magnética.

Exemplo 10.1

Calcule o espaço gasto para armazenar em fita magnética um arquivo A contendo 10.000 RL de 100 bytes cada um. O armazenamento será realizado com FB = 10, densidade de gravação de 1600 bpi.

Sabe-se que 1 gap ocupa 0,75″, que cada campo de controle de bloco gasta 200 bytes e que 1 pé = 12 polegadas.

Solução

Arquivo A = 10.000 RL   1 RL = 100 bytes   FB = 10

Controle = 200 bytes   Densidade = 1600 bpi

← Bloco →

Controle	RL1	RL2	RL3	RL4	RL5	RL6	RL7	RL8	RL9	RL10

200 bytes 100 100 100

Tamanho de 1 bloco: 200 + 10 × 100 = 1200 bytes

Quantidade de blocos: FB = NRL/NRF. Assim, NRF = NRL/FB = 10.000/10 = 1000 RF

Espaço gasto com gaps: 1000 gaps (= NRF) × 0,75″ = 750 polegadas

Espaço gasto com blocos: 1000 blocos × 1200 bytes = 1.200.000 bytes

Como a densidade é 1600 bpi, então: 1.200.000/1600 = 750 polegadas

Total do espaço com o arquivo: 750″ (gaps) + 750″ (blocos) = 1500 polegadas

1500/12 = 125 pés

10.3.5 Discos Magnéticos

Discos magnéticos (também disquetes, fitas magnéticas, CD-ROM) são componentes de um computador que podem ser enquadrados em duas áreas distintas: como memória secundária e como dispositivos periféricos de E/S. Atualmente, esses dispositivos são largamente utilizados seguindo os dois pontos de vista. Neste item, vamos descrever as principais características que definem um disco magnético, tais como: a organização de armazenamento de dados e a base de seu funcionamento.

Estaremos nos referindo aqui a discos magnéticos de um modo geral, sejam disquetes ou discos rígidos com tecnologia Winchester. Mais adiante, serão abordados alguns pontos específicos destes dois tipos de discos, os mais empregados em microcomputação.

O disco magnético (disco rígido ou hard-disk) é uma superfície circular, fina e coberta com uma camada de material magnetizável. O material pode estar presente em uma ou em ambas as superfícies do disco, normalmente chamadas faces. Atualmente qualquer disco magnético para sistemas de computação é fabricado com dupla face de armazenamento de modo que tenha maior capacidade de armazenamento. A Fig. 10.31 mostra um exemplo de um tipo de disco magnético, muito empregado em microcomputadores.

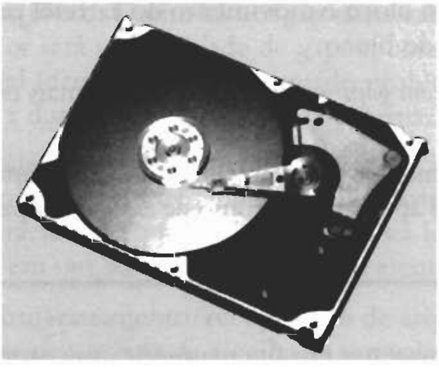


Figura 10.31 Disco magnético utilizado em microcomputadores.

A Fig. 10.32 mostra o esquema básico de um disco magnético rígido, aparecendo a superfície magnetizável, constituída de áreas circulares concêntricas, denominadas trilhas, onde são armazenados os bits de informação. Cada bit é representado por um campo magnético, cuja direção define o valor do bit, 0 ou 1. O disco gira constantemente em torno do seu eixo central de modo idêntico ao da rotação de um disco de som.

Sobre a superfície, um elemento mecânico (braço) transporta a cabeça de leitura/gravação, efetuando um movimento transversal sobre as trilhas, de modo a realizar as operações de leitura e gravação sobre cada trilha.

Todas as trilhas armazenam a mesma quantidade de bytes; isto é possível devido à diferença de densidade de gravação entre a trilha mais externa e a mais interna.

As N trilhas são numeradas (endereços) a partir da trilha mais externa (trilha de endereço 0) até a trilha mais interna (trilha de endereço N − 1); como, em geral, cada trilha possui uma grande capacidade de armazenamento de bytes, costuma-se dividir a trilha em pedaços menores, denominados *setores* (ver Fig. 10.32), os quais servem de unidade de transferência de dados.

Na prática, há diferentes métodos de acesso aos discos, bem como diversas unidades de transferência, dependendo do sistema operacional, do próprio sistema de disco, etc. Há sistemas que efetuam transferência de dados disco/UCP-MP setor por setor; outros, devido à maior capacidade dos discos e ao volume de transferências, movimentam um grupo de setores (chamado cluster, denominação utilizada pela Microsoft nos sistemas operacionais DOS e Windows) de cada vez, embora a unidade básica de armazenamento continue a ser o setor. A IBM utiliza, em seus sistemas de grande porte, o bloco (conjunto de RL) ou cilindro como unidade de armazenamento e transferência.

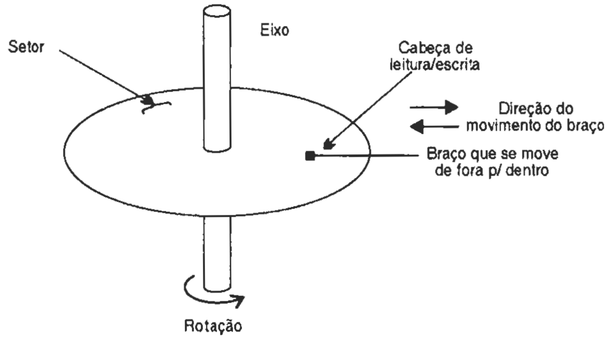

Setor
Eixo
Cabeça de leitura/escrita
Direção do movimento do braço
Braço que se move de fora p/ dentro
Rotação

Figura 10.32 Diagrama esquemático de um disco magnético.

O *tempo de acesso* em disco é o período gasto entre a ordem de acesso (com o respectivo endereço) e o final da transferência dos bits. Esse tempo é o somatório de três tempos menores:

a) *Tempo de SEEK (busca)* — gasto para interpretação do endereço pela unidade de controle e movimento mecânico do braço para cima da trilha desejada. É o maior componente do tempo de acesso.

b) *Tempo de latência* — período decorrido entre a chegada da cabeça de leitura e gravação sobre a trilha e a passagem do bloco (setor) desejada sobre a referida cabeça (depende da velocidade de rotação do disco).

c) *Tempo de transferência* — gasto para a efetiva transmissão dos sinais elétricos (bits) para o destinatário.

Há certos dispositivos, em geral discos rígidos de maior capacidade de armazenamento, que são constituídos de vários discos superpostos, conforme mostrado nas Figs. 10.33 e 10.34. O conjunto é armado de forma integrada, sendo cada superfície associada a um braço específico, com sua própria cabeça de leitura/gravação. O conjunto formado por todos os braços se move por igual, acionado através de um mecanismo atuador único.

Como os braços/cabeças se movimentam juntos, quando o atuador se desloca para acessar uma determinada trilha de certa superfície (o endereço de acesso pode ser, por exemplo, trilha 12 da superfície 05), todas as cabeças estacionam sobre a trilha de mesmo endereço (trilha 12, nesse exemplo) em cada superfície. O con-

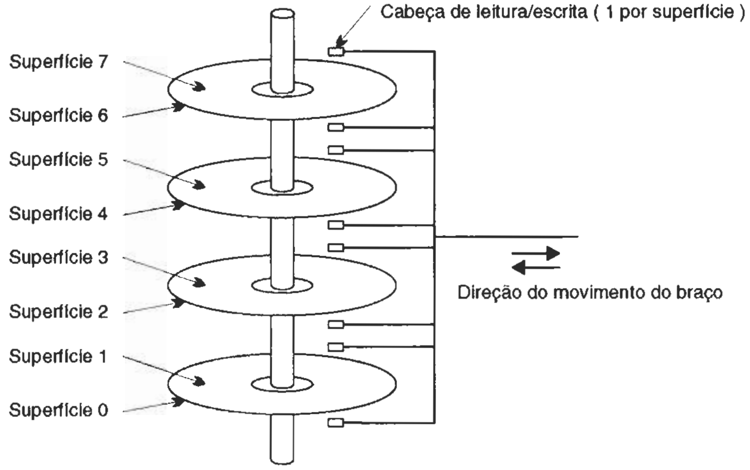

Cabeça de leitura/escrita (1 por superfície)
Superfície 7
Superfície 6
Superfície 5
Superfície 4
Superfície 3
Superfície 2
Superfície 1
Superfície 0
Direção do movimento do braço

Figura 10.33 Esquema de um disco magnético com várias superfícies superpostas em um único eixo. Cada superfície é percorrida por um braço com cabeça de leitura/escrita e todos os braços estão presos a um único mecanismo de movimento.

junto de trilhas de mesmo endereço, acessado em um único movimento do atuador, denomina-se *cilindro*, podendo também servir como unidade de transferência em certos sistemas.

O acesso por cilindro aumenta a produtividade do sistema de disco, quando se movimentam grandes volumes de dados, visto que se economiza tempo de busca (de seek) comparativamente ao processo de armazenamento trilha por trilha.

Se, por exemplo, desejarmos armazenar em disco um arquivo que consumirá um espaço correspondente a 200 trilhas, serão gastos 200 tempos de busca (de seek) no caso de a organização dos endereços ser por trilha. No entanto, se o sistema de disco for organizado de modo a armazenar e recuperar dados cilindro a cilindro, poderemos verificar que o tempo total de busca será bem menor.

Assim, se no exemplo dado o disco possuir 20 superfícies magnetizáveis, cada cilindro compreenderá 20 trilhas (o cilindro 23 é constituído de todas as 20 trilhas de endereço 23, uma para cada superfície); nesse caso, seriam consumidos apenas 10 tempos de busca (e não mais os 200 anteriores), um para cada cilindro.

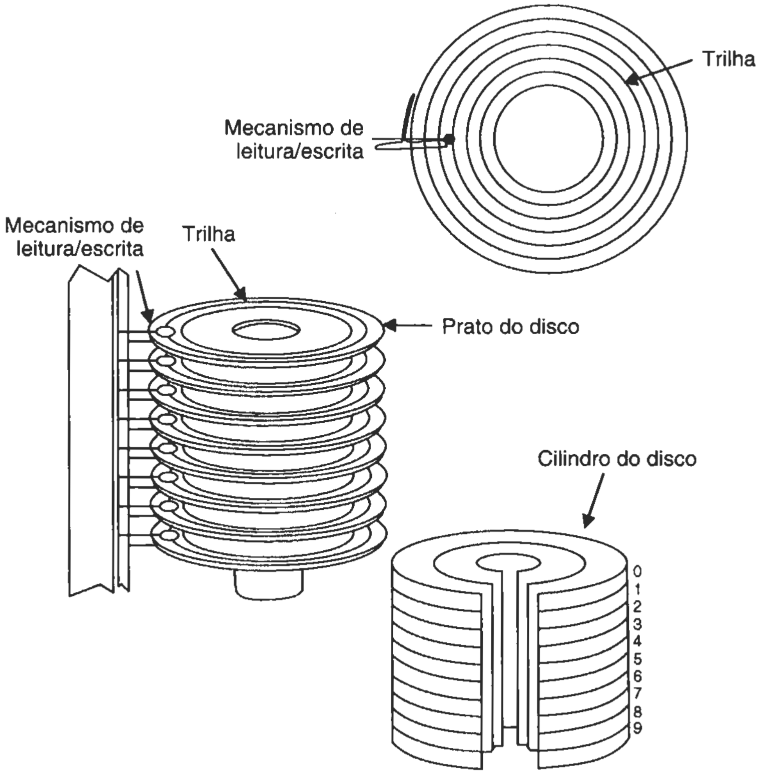

Trilha
Mecanismo de leitura/escrita
Mecanismo de leitura/escrita
Trilha
Prato do disco
Cilindro do disco
0
1
2
3
4
5
6
7
8
9

Figura 10.34 Exemplo de organização de um disco magnético com múltiplos pratos.

10.3.5.1 Discos Flexíveis ou Disquetes (Floppy-Disk)

As unidades de disquetes (*floppy-disk drivers*) possuem características semelhantes às das unidades de disco rígido, de maior capacidade. A diferença entre os dois dispositivos reside mais na capacidade de armazenamento, na velocidade de acesso e no tempo de transferência de informações.

As unidades de disquetes (e os próprios disquetes) foram desenvolvidas na década de 1960 como uma alternativa mais barata para os sistemas de disco. Porém, esse meio se tornou mais popular, crescendo enormemente sua utilização pelos usuários, com o surgimento dos microcomputadores e a conseqüente demanda por um periférico mais rápido e versátil que os cassetes (primeiros dispositivos de armazenamento secundário usados nos microcomputadores).

O disquete é um meio de armazenamento removível, o que permite o transporte de informação de um sistema para outro. As informações são armazenadas de forma idêntica à dos discos rígidos, isto é, em trilhas e setores de cada superfície (um disquete possui as duas superfícies magnetizáveis, embora apenas uma delas o fosse nos primeiros disquetes). O acionador de disquete em um computador (drive) contém um mecanismo de leitura/gravação para cada superfície. Os disquetes possuem ainda uma abertura para que a cabeça de leitura/escrita tenha acesso às trilhas/setores (nos atuais disquetes de 3,5″ esta abertura fica coberta por um elemento metálico).

Em geral, quando um usuário se dispõe a adquirir um sistema de disco, ele procura comparar os diversos modelos oferecidos através de três fatores principais:

- a capacidade total de armazenamento do meio;
- a taxa de transferência de bits (quantos bits/s); e
- o tempo médio de acesso.

Os primeiros disquetes (sistema IBM 3740) possuíam um tamanho de 8″ (o mercado de disquetes costuma identificar esses dispositivos pelo tamanho do elemento de armazenamento) e capacidade de armazenamento de até 360 Kbytes. Mais tarde, com os sistemas do tipo IBM PC surgiram os disquetes de 5 1/4″, face simples e dupla face de armazenamento (2 cabeças de leitura/gravação e, conseqüentemente, cilindro com 2 trilhas cada um), maior capacidade e velocidade.

Com o lançamento dos microcomputadores da linha PS/2, a IBM também implantou um novo sistema de disquete, com invólucro mais rígido que os modelos anteriores (flexíveis), 3 1/2″ de tamanho e maior capacidade de armazenamento. Atualmente, podem ser encontrados disquetes de 3 1/2″ com 1.44Mb, com capacidade de 2.88Mb e até de 120Mb.

A taxa de transferência de unidades de disquetes não é tão grande se comparada com sistemas de disco rígido (disquetes podem alcançar taxas de transferência da ordem de centenas de Kbytes/s, enquanto os discos rígidos podem transferir alguns Mb/s).

O tempo médio de acesso também é relativamente elevado nas unidades de disquetes (da ordem de 60 a 100 ms), ao passo que unidades de disco rígido podem realizar acessos com tempos bem menores que 60 ms (atualmente, fabricantes típicos, como a Seagate, a Western Digital, a Fujitsu e outros, vêm produzindo unidades de disco rígido para microcomputadores com tempos de acesso menores que 10 ms).

Os atuais sistemas de computação, por terem capacidade de armazenamento secundário cada vez maior (discos rígidos de vários Gigabytes de capacidade), utilizam menos os disquetes, por esses possuírem muito menos capacidade de armazenamento. Em contrapartida, os CD-ROM e discos tipo zip (zip disks, que são manipulados por unidades denominadas zip drivers) têm tido uma aceitação cada vez maior, sendo que as unidades leitoras de CD-ROM já fazem parte integrante de praticamente todos os computadores vendidos atualmente pelo mercado.

No item 10.5 serão abordados os aspectos mais relevantes desses dispositivos.

10.3.5.2 Unidades de Disco doTipo Winchester

No início da década de 1980, justamente com a explosão de demanda por microcomputadores, tanto para uso comercial quanto pessoal, cresceu também a necessidade de sistemas de armazenamento de maior capacidade que os disquetes e, principalmente, com maiores taxas de transferência e menores tempos de acesso. Os discos rígidos de sistemas de grande porte eram caros e de tamanho físico incompatível com o tipo de ambiente dos microcomputadores onde deveriam ser inseridos.

A IBM desenvolveu, então, uma tecnologia de fabricação de discos rígidos, de tamanho pequeno e compacto, baixo custo e desempenho elevado para o padrão dos sistemas de microcomputadores existentes. Esta tecnologia foi chamada de Winchester, embora o nome esteja relacionado (segundo alguns) ao famoso fuzil 30-30 Winchester. O fato é que a técnica ficou tão popular que todas as unidades de disco para micros vêm sendo fabricadas com essa tecnologia; como tem ocorrido com outros produtos comerciais de sucesso, o nome da tecnologia de fabricação se confundiu com o da própria unidade. É costume se dizer: "vou adquirir um microcomputador com Winchester de 60Mb", em vez de explicitar: "vou adquirir um microcomputador com unidade de disco rígido de 60Mb, do tipo Winchester".

Como nas unidades de disquete o espaçamento entre as trilhas é relativamente grande (por ser um dispositivo de relativamente baixa precisão) e a densidade de gravação também não é acentuada, esses elementos possuem reduzida capacidade de armazenamento.

Para aumentar a capacidade, reduzindo também o tempo de acesso, os sistemas de disco rígido precisam ter características de fabricação mais precisas; a densidade de gravação deve aumentar e o espaço entre trilhas deve diminuir. Para isso foi necessário desenvolver métodos e técnicas mais elaboradas, de modo a assegurar que a cabeça de leitura/gravação se posicione exatamente em cima da trilha desejada (o menor desalinhamento acarretaria leituras erradas devidas à proximidade entre as trilhas).

A tecnologia Winchester procurou solucionar o problema do posicionamento da cabeça, tornando o sistema disco/cabeça/atuador uma unidade integrada e selada. Com isso, deixou de haver necessidade de alinhamento (mais precisão) e se pôde aumentar a densidade de gravação e a quantidade de trilhas, obtendo-se assim maior capacidade de armazenamento.

Os primeiros discos rígidos tipo Winchester surgiram no mercado com capacidade de armazenamento de 5Mb, logo aparecendo discos com capacidade de 10Mb e 20Mb. A demanda por maior espaço de armazenamento secundário continua a crescer rapidamente com o aparecimento de programas de aplicação gráfica, logo surgindo discos com capacidade de armazenamento de centenas de megabytes. Atualmente há sistemas de microcomputadores sendo oferecidos no mercado com discos rígidos de capacidade da ordem de 1.2Gb a 100Gb e até maiores.

10.3.5.3 Cálculo de Espaço de Armazenamento em Discos

O problema do cálculo de espaço em disco necessário para o armazenamento de arquivos é semelhante ao que foi mostrado para o cálculo de espaço em fitas magnéticas. Uma das possíveis diferenças consiste, em grande parte dos sistemas, na adoção do setor (ou grupo de setores) como unidade de transferência fixa e, portanto, deixa de existir *o fator de bloco* variável.

No caso de discos, é necessário calcular quantas trilhas (ou cilindros) serão consumidas por um arquivo constituído de N registros lógicos. Deve-se considerar a divisão das trilhas em setores, cada um com uma quantidade fixa de bytes para armazenamento.

Exemplo 10.2

Deseja-se saber qual será o dispêndio de espaço para armazenar em disco um arquivo com 1000 RL de 80 bytes cada um. O disco possui 40 trilhas com 9 setores de 512 bytes para dados em cada um.

Solução

Total de bytes do arquivo: $1000 \times 80 = 80.000$ bytes

Quantidade de setores necessária: $80.000/512 = 156{,}25 = 157$ setores

Quantidade de trilhas: $157/9 = 17{,}4 = 18$ trilhas

Exemplo 10.3

Um sistema de armazenamento em disco magnético possui discos com 5 superfícies de dupla face, todas com as respectivas cabeças de leitura/gravação. Cada superfície contém 115 trilhas, cada uma com possibilidade de gravar 9500 bytes de dados (e informações de controle). Podem ser armazenados nesse disco blocos de tamanho variável, de acordo com o fator de bloco (H) escolhido; cada bloco contém 80 bytes para informações de controle. Em cada trilha somente podem ser gravadas quantidades inteiras de blocos (ou RF), não sendo possível gravar parte de um bloco em uma trilha e o restante em outra. Calcule a quantidade de cilindros a ser consumida com o armazenamento de um arquivo constituído de 8000 RL de 110 bytes cada, empregando-se um FB de 12.

Solução

Tamanho de 1 bloco: 12 × 110 = 1320 bytes (se o FB = 12, então: 12 RL em cada bloco)

Quantidade de blocos por trilha: 9500/1320 = 7,2 = 7 blocos (sobram 260 bytes em cada trilha)

Quantidade de blocos necessária: 8000/12 = 666,66 = 667 blocos

Se FB = NRL/NRF, então:

NRF = NRL/FB

Quantidade de trilhas: 667/7 = 95,29 = 96 trilhas (porque são 7 blocos/trilha)

Como o disco possui 5 pratos de dupla face, há 10 superfícies de armazenamento; há, então, cilindros com 10 trilhas em cada um (o cilindro 3, por exemplo, compreende todas as trilhas 3 das 10 superfícies).

O disco possui um total de 115 cilindros.

Cálculo da quantidade necessária de cilindros: 96/10 = 9,6 = 10 cilindros.

10.3.6 Mouse

Em essência, o mouse é um dos dispositivos de entrada de um sistema de computação cujo propósito é facilitar o trabalho do usuário em sua comunicação com o sistema. Em vez de o usuário ser obrigado a digitar comandos que precisam ser aprendidos, decorados ou lidos em algum lugar, com o mouse o usuário necessita somente de um pouco de coordenação motora para movimentar o dispositivo. A Fig. 10.35 mostra um modelo de mouse.

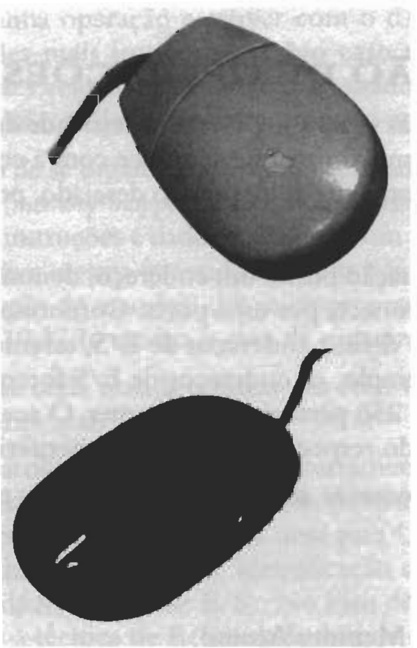


Figura 10.35 Mouse.

É evidente que, sendo um dispositivo basicamente apontador (na realidade, o usuário movimenta o mouse para apontar para alguma figura — chama-se ícone em uma tela gráfica), ele é apropriado apenas como elemento de interligação visual do usuário com o sistema.

No que se refere ao funcionamento de um mouse (assim chamado em razão de seu formato pequeno, sua ligação ao sistema por um fio que pode parecer um rabo de rato e seus movimentos agitados na mão do usuário, fazendo lembrar — para quem criou o nome, pelo menos — um ratinho), pode-se descrevê-lo de modo simples.

O dispositivo possui um sensor (há três tipos de sensores: *mecânicos, óticos* e *ótico-mecânicos*) que capta seu movimento em uma superfície plana e transmite informações sobre este movimento ao sistema de computação e dois ou três botões. Um programa que controla o funcionamento do dispositivo e se comunica com o computador (é parte do interface) converte as informações de movimento do mouse em movimento de um elemento apontador na tela de vídeo. O usuário escolhe o que quer apontar e seleciona, pressionando um dos botões do mouse.

Mouses mecânicos costumam usar uma esfera coberta com borracha (para deslizar melhor) que gira acompanhando o movimento do mouse. O movimento de rotação da esfera é transmitido a dois rodetes perpendicularmente colocados e que possuem rodas com contatos de metal.

À medida que as rodas giram (devido ao movimento do mouse pela superfície), os contatos tocam periodicamente escovas colocadas no interior do mouse, completando o circuito. Embora simples e barato, este projeto é ineficaz porque volta e meia o mouse tem problemas de fechamento dos contatos devido à poeira e à sujeira. Os sistemas ótico-mecânicos são mais utilizados porque combinam bem as melhores características de dispositivos puramente mecânicos e óticos, sem absorver seus defeitos.

Os mouses do tipo *ótico-mecânico* possuem o mesmo mecanismo de esfera e rolete que os dispositivos mecânicos, exceto que os roletes são conectados a rodas vazadas (com furos). À medida que as rodas giram, elas ora bloqueiam, ora permitem a passagem de luz, que é produzida por um LED, e estas transições são detectadas por semicondutores sensíveis à luz. Este tipo tem a vantagem (como os mecânicos) de poder funcionar em qualquer superfície.

A maioria dos mouses atualmente fabricados possui uma resolução de 400 pontos por polegada, que significa que foram registrados 400 sinais de movimento do mouse, enquanto ele andou 1 polegada.

Além do mouse, tem se usado também um outro tipo de dispositivo apontador, a *track-ball*, popularizada com o aparecimento de laptops e notebooks, por não requerer movimento em uma superfície. É apenas a esfera que gira, movimentando o curso na tela do vídeo.

10.4 MÉTODOS DE REALIZAÇÃO DE OPERAÇÕES DE E/S

Durante a execução de um programa, por diversas vezes a UCP tem necessidade de enviar ou receber dados de algum dispositivo periférico. Para tanto, da mesma forma que acontece com a comunicação UCP/MP, é necessário que a UCP indique o endereço correspondente ao periférico desejado, pois há sempre mais de um periférico ligado a um sistema de computação.

Cada periférico conectado ao sistema de computação possui um endereço, denominado endereço da porta de E/S (*port. number*), visto que cada periférico se conecta por uma porta. Conforme a quantidade de bits que tenha sido estabelecida no projeto do sistema para definir endereços de E/S, teremos o limite de periféricos que podem ser conectados ao sistema. Se, por exemplo, os endereços de E/S forem números com 8 bits de tamanho, então somente será possível conectar até 256 periféricos ao sistema. O acesso da UCP a um periférico é obtido através do barramento do sistema e do respectivo interface do periférico.

A comunicação entre UCP e o interface pode ocorrer através de um entre três possíveis métodos:

1. Entrada/saída por programa.
2. Entrada/saída com emprego de interrupção.
3. Acesso Direto à Memória (DMA — Direct Memory Access).

No primeiro caso, a UCP controla diretamente todas as etapas da comunicação, executando as instruções necessárias, tais como enviar um dado para o interface (por exemplo, a instrução assembly OUT ou OUTS, dos processadores Intel), ler dados do interface (as instruções IN ou INS), verificar o estado do periférico e outras. Considerando que os periféricos transferem dados em um tempo muito maior do que a UCP gasta para processar uma soma, por exemplo, este processo é altamente ineficiente.

Uma alternativa a este método consiste no emprego da técnica de interrupção. Neste caso, a UCP comanda, por uma instrução, o início de uma operação de entrada ou de saída e passa a realizar outras tarefas até que o periférico sinalize (por um sinal de interrupção — ver item 10.4.2) que a operação terminou. Com isso, a UCP não perde tempo verificando se a operação do periférico terminou.

Mais ainda assim a UCP participa do processo e, com isso, perde tempo que poderia estar utilizando para realizar um processamento efetivo, como alguma operação matemática. Por essa razão, há um terceiro método pelo qual a transferência de dados de/para o interface se realiza entre este e a memória, sem interveniência do processador. Trata-se do acesso direto à memória ou DMA.

10.4.1 Entrada/Saída por Programa

Com esse método, a UCP executa diretamente instruções de E/S, enviando e recebendo dados do interface de E/S. Cada instrução serve para uma ação típica de E/S, como examinar o estado de um interface, realizar a operação de E/S com o interface (enviar um byte de dados ou receber um byte de dados). Uma típica operação de entrada (E) em um microcomputador é apresentada no fluxograma da Fig. 10.36.

O fluxograma considera que um conjunto de bytes ou palavras deve ser introduzido no sistema (operação de entrada) e que cada byte ou palavra é trazido para a UCP, sendo modificado de alguma forma e, em seguida, transferido para um buffer de memória. Quando todo o conjunto de dados tiver sido armazenado no buffer, este será processado.

Para executar uma operação de E/S, a UCP deve enviar o endereço do dispositivo e do seu interface e o comando de E/S desejado. Um comando de E/S pode ser de leitura (o interface interpreta o comando e busca um byte ou palavra do registrador interno do dispositivo e o coloca em seu registrador). A UCP necessita posteriormente receber o dado, solicitando que o interface o coloque no barramento (outra instrução). Outro possível comando de E/S é o de escrita, semelhante ao de leitura, exceto que em sentido contrário. Há também comandos de verificação de erros e de controle do periférico.

Conforme já mencionado, a grande desvantagem deste método consiste no intenso uso de UCP. Como tem que manter um loop de execução para diversas atividades (por exemplo, verificar o estado de um interface, se este completar uma operação e estiver com o dado disponível), ocorre um desperdício de uso em detrimento de atividades mais importantes. No caso do uso deste método de E/S, uma solução para compatibilizar as diferentes velocidades entre o processador e o periférico é justamente o loop de interrogação do estado do periférico, não se devendo enviar o dado enquanto ele não estiver pronto para recebê-lo.

Há duas possibilidades de se organizar a comunicação entre UCP, memória principal e interface de E/S: por memória compartilhada (*memory-mapped*) e E/S isolada (*isolated I/O*). No primeiro caso, a memória principal é compartilhada tanto por instruções e dados comuns de um programa quanto por instruções/dados de operações de E/S. Os endereços têm a mesma quantidade de bits e as instruções o mesmo formato. A Fig. 10.37(a) mostra um exemplo de organização de memória dos microprocessadores 8086/8088 para o caso de memória compartilhada, enquanto a Fig. 10.37(b) mostra o caso de organização isolada de E/S (isto é, isolada da memória).

O processo de entrada/saída isolada consiste em criar um espaço de memória próprio de E/S e diferente, portanto, da memória principal.

Com memória compartilhada, basta haver no barramento de controle uma única linha de leitura (*read*) e de escrita (*write*), visto que a memória é a mesma e, portanto, somente poderá ocorrer em cada instante uma operação de leitura/escrita para UCP/MP ou uma para UCP/E/S. No caso de E/S isolada, isto não é possível. É, pois, necessário haver um sinal de identificação do topo de operação sempre que um endereço for colocado na linha (se é da MP ou se de E/S). No caso de microprocessadores da família Intel, por exemplo, usa-se mais comumente a técnica de E/S isolada do que a de memória compartilhada.

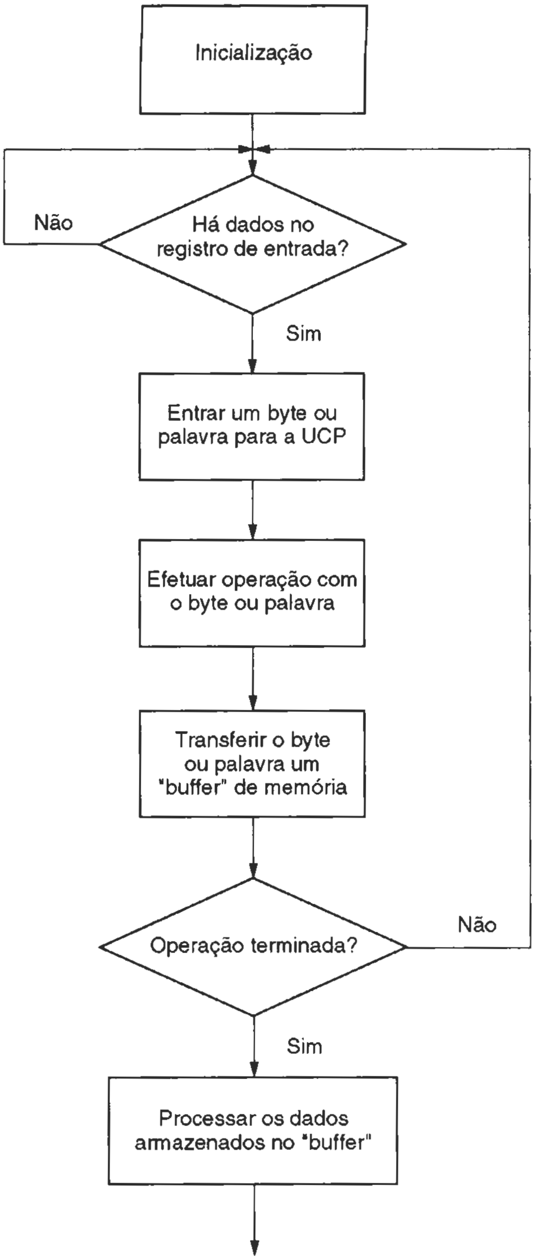
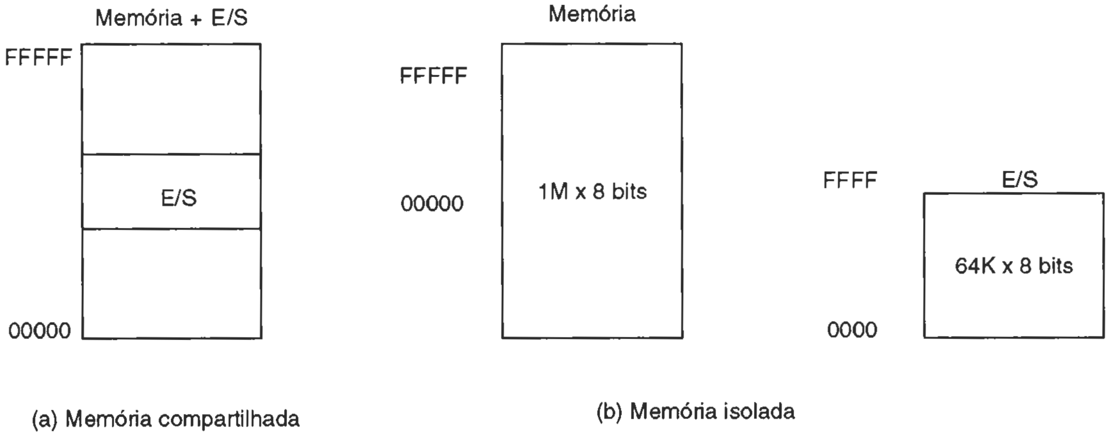

Inicialização
Não
Há dados no registro de entrada?
Sim
Entrar um byte ou palavra para a UCP
Efetuar operação com o byte ou palavra
Transferir o byte ou palavra um "buffer" de memória
Não
Operação terminada?
Sim
Processar os dados armazenados no "buffer"

Figura 10.36 Fluxograma de uma típica operação de entrada de dados em um microcomputador.


Memória + E/S
FFFFF
E/S
00000
Memória
FFFFF
1M x 8 bits
00000
FFFF
E/S
64K x 8 bits
0000

(a) Memória compartilhada

(b) Memória isolada

Figura 10.37 Exemplo de organização de memória nos microprocessadores Intel 8086/8088.

Os endereços de E/S isolada são chamados portas (*ports*). No caso desses microprocessadores, há em geral uma convenção estabelecendo que os endereços de E/S (diferentes dos da MP) podem ter 8 bits ou 16 bits. Com 8 bits são endereçados dispositivos localizados na própria placa principal, como o controlador de tempo (*timer*) e o interface do teclado, e com 16 bits são endereçados os dispositivos externos, como vídeo e controladores de disco.

O método de E/S isolada tem a vantagem de não utilizar espaço da memória principal, deixando-a toda para outras aplicações, mas tem a desvantagem de só ser utilizado com instruções especiais de E/S (no caso de microprocessadores Intel, são usadas apenas instruções IN, INS, OUT e OUTS). Além disso, também foram desenvolvidos sinais especiais de controle para o espaço de E/S (um para leitura e outro para escrita). Por outro lado, o método de memória compartilhada (*memory-mapped*) tem a vantagem de não necessitar de qualquer instrução especial, ou seja, qualquer instrução que utilize a memória principal se aplica também a E/S. Mas tem a desvantagem de ocupar parte do espaço de memória principal para uso de E/S.

10.4.2 Entrada e Saída com Emprego de Interrupção

O método de realizar E/S através da contínua atenção e controle da UCP é bastante ineficiente, principalmente devido à lentidão dos dispositivos de E/S em relação à velocidade da UCP.

Uma alternativa válida para evitar a perda de tempo de a UCP continuamente interrogar o estado de um periférico consiste em utilizar uma técnica denominada *interrupção*. Nesse caso, a comunicação UCP/interface/periférico funciona da seguinte maneira:

a) A UCP emite a instrução de E/S para o interface e, como não deverá haver uma resposta imediata, em vez de ficar continuamente verificando o estado do periférico, a UCP desvia-se para realizar outra atividade (provavelmente executar um outro programa, suspendendo a execução daquele programa que requer a E/S).

b) Quando o interface está pronto para enviar os dados do periférico para a UCP, ela "avisa" a UCP por meio de um sinal de interrupção. Chama-se *interrupção* porque realmente o sinal interrompe a atividade corrente da UCP para que esta dê atenção ao dispositivo que a interrompeu.

c) A UCP inicia, então, o programa de E/S como se fosse o método anterior.

Para que o método seja mais bem assimilado, é interessante que o conceito de interrupção seja compreendido um pouco mais.

Interrupção consiste em uma série de procedimentos que suspendem o funcionamento corrente da UCP, desviando sua atenção para outra atividade. Quando esta outra atividade é concluída, a UCP retorna à execução anterior do ponto onde estava antes de ser interrompida (é possível que a UCP, após concluir a atividade que a interrompeu, não retorne imediatamente ao programa interrompido, fazendo isto um pouco mais tarde).

A Fig. 10.38 mostra um exemplo de interrupção e seu funcionamento básico, e a Fig. 10.39 ilustra um barramento com uma linha de interrupção. Todo sistema possui, pelo menos, uma destas linhas, por onde circula o sinal de interrupção que desencadeia a suspensão de atividade corrente da UCP.

Há basicamente duas classes de interrupção:

- *interrupções internas* ou de programas (às vezes chamadas de *traps* ou *exception*);
- *interrupções externas.*

As interrupções internas a um programa ocorrem devido a algum tipo de evento gerado pela execução de uma instrução ou até mesmo programado. Pode ser, por exemplo, uma divisão por zero, um overflow em uma operação aritmética, um código de operação inválido, etc.

As interrupções externas são causadas por um sinal externo à UCP que a interrompe. Em geral, isto está relacionado a um interface de E/S que pretende "avisar" à UCP que um determinado periférico deseja atenção para transferir dados.

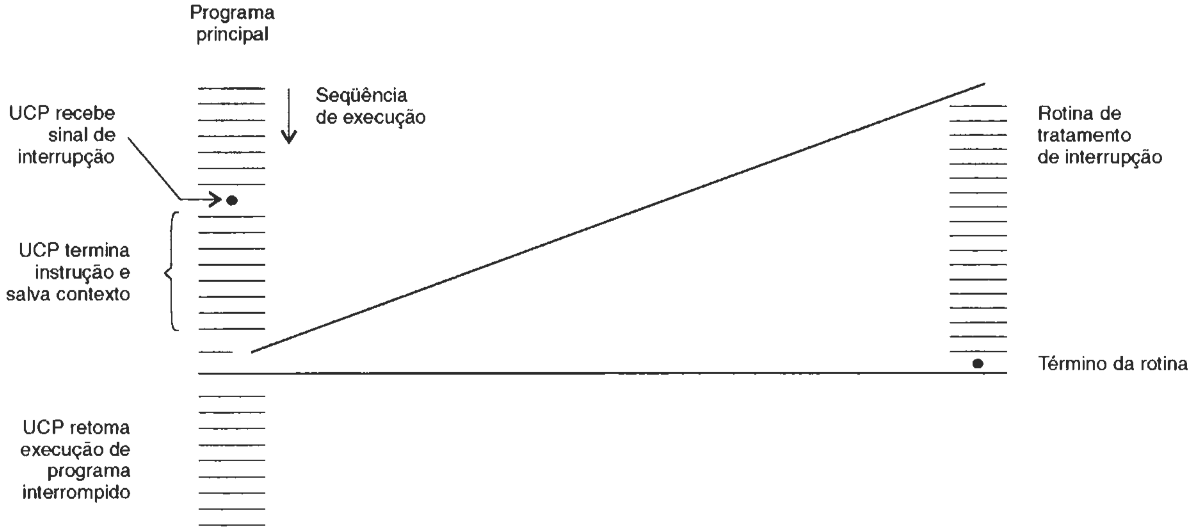

Programa
principal
Seqüência
de execução
UCP recebe
sinal de
interrupção
UCP termina
instrução e
salva contexto
UCP retoma
execução de
programa
interrompido
Rotina de
tratamento
de interrupção
Término da rotina

Figura 10.38 Exemplo de uma seqüência de atividades que ocorrem durante o desenrolar de uma interrupção.

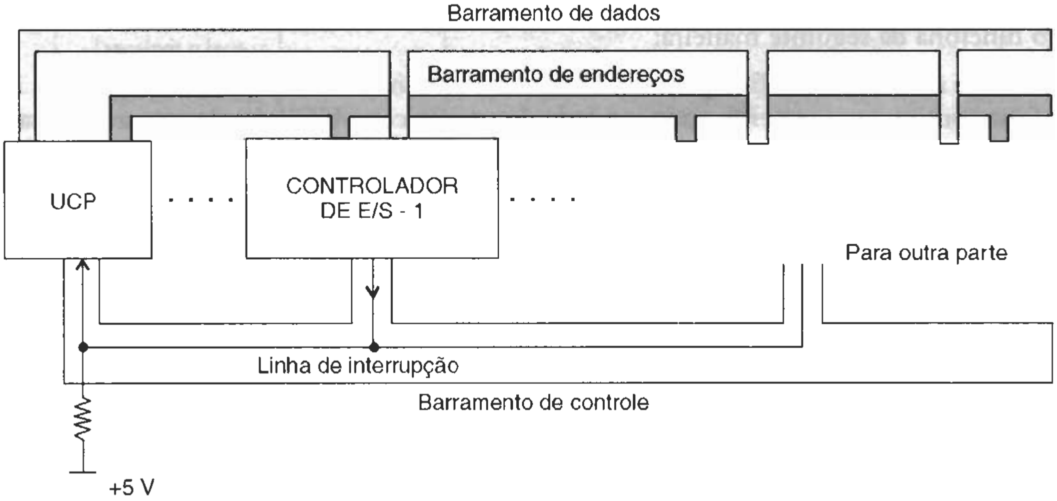

Barramento de dados
Barramento de endereços
UCP
CONTROLADOR
DE E/S - 1
Para outra parte
Linha de interrupção
Barramento de controle
+5 V

Figura 10.39 Tratamento de uma interrupção (elementos que participam do processo).

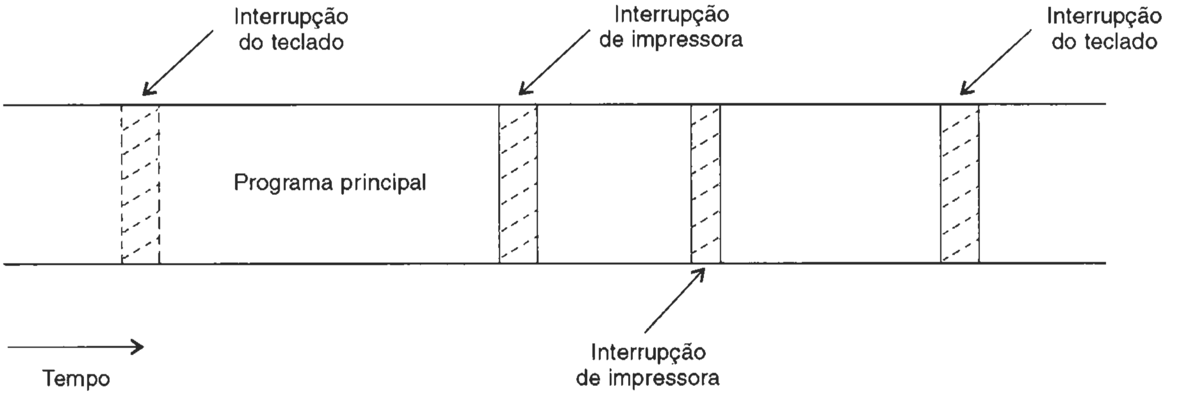

Interrupção
do teclado
Interrupção
de impressora
Interrupção
do teclado
Programa principal
Interrupção
de impressora
Tempo

Figura 10.40 Exemplo de procedimentos de interrupção juntamente com a execução de programa (distribuição de tempo entre os diversos eventos).

A Fig. 10.40 mostra, em uma escala de tempo, a relação no tempo entre eventos que ocorrem em um sistema, indicando um digitador utilizando um teclado (cada tecla acarreta uma interrupção para que o caractere seja lido), uma impressora removendo dados da memória e um programa sendo executado.

O programa é o programa principal, que é interrompido a cada pressionamento de uma tecla, bem como a cada caractere transmitido para a impressora. Observe a curta duração de cada procedimento de interrupção em comparação com o tempo gasto durante a execução do programa (entre as interrupções).

Ao se efetivar a interrupção da UCP, algumas ações devem ser tomadas:

- Identificação do tipo de interrupção (de que se trata?).
- Qual o dispositivo que sinalizou?
- Como reagir à ocorrência?
- Dar atenção imediata? Aguardar para mais tarde? Ignorar?
- O que deve acontecer com o programa interrompido? Quando ele retornará à sua execução?

Para atender a quase todas essas questões, o sinal de interrupção acarreta o desvio na seqüência de execução do programa corrente (ver Fig. 10.38), passando a ser executada uma rotina usualmente denominada *Rotina de Tratamento de Interrupção* (*Interruption Handler*).

No entanto, antes de esse desvio ocorrer, a UCP termina a instrução corrente e executa uma ação denominada, de um modo geral, *salvamento do contexto* do programa corrente. O fato é que os registradores da UCP estão, no momento da interrupção, armazenando valores usados pelo programa que está sendo interrompido e que continuarão a ser utilizados quando o programa retornar à sua execução. Por outro lado, o novo programa a ser executado vai precisar utilizar os mesmos registradores.

Além disso, o CI (Contador de Interação) contém o endereço da próxima instrução do programa que está sendo interrompido, o qual precisa ser também guardado para ser utilizado quando o programa retomar sua execução. Este conjunto de dados e endereços do programa corrente chama-se *contexto* e deve ser armazenado em uma área previamente designada para isto na memória principal. Após o salvamento deste contexto, o CI recebe o endereço inicial da rotina de tratamento de interrupção e sua execução é iniciada.

Voltando ao problema inicial, podemos observar que esta modalidade de realizar operações de E/S melhorou o desempenho dos sistemas, mas ainda apresenta algumas desvantagens. Embora a UCP não precise mais interrogar o estado de disponibilidade de um periférico, ela continua gastando tempo para executar o programa de E/S para efetivar a transferência dos dados.

Um interface muito utilizado com este método denomina-se, em inglês, Programmable Peripheral Interface, fabricado pela Intel com o código 82C55A.

10.4.3 Acesso Direto à Memória — DMA

A melhor alternativa para se realizar operações de E/S com o máximo de rendimento da UCP é o método denominado acesso direto à memória (DMA — Direct Memory Access).

De modo geral, a técnica DMA consiste na realização da transferência de dados entre um determinado interface e a memória principal, praticamente sem intervenção da UCP. Esta se limita a solicitar a transferência para um dispositivo denominado controlador de acesso direto à memória (*DMA controller*), o qual realiza por si só a transferência. A UCP fica liberada para realizar outras atividades. Quando o controlador termina a transferência, ele sinaliza para a UCP através de uma interrupção.

Na realidade, o controlador de DMA age como um mestre ou estação principal para controlar o barramento (ver item 6.5.3) quando entra em funcionamento. A Fig. 10.41 mostra um exemplo de entrada/saída com emprego de DMA.

Durante o funcionamento normal da UCP, a chave de barramento número 1 está ligada, permitindo a passagem de endereços/dados entre UCP e MP, enquanto as chaves de barramento 2 e 3 estão desativadas, não permitindo passagem de informações. Quando o controlador DMA entra em funcionamento, a chave 1 e as chaves 2 e 3 são ligadas.

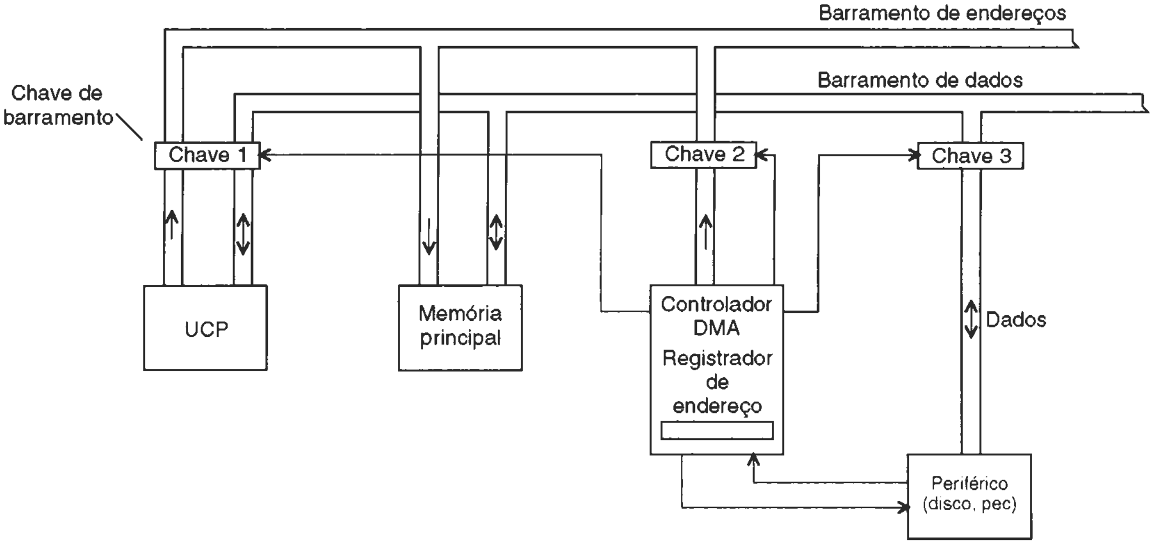

Barramento de endereços
Barramento de dados
Chave de barramento
Chave 1
Chave 2
Chave 3
UCP
Memória principal
Controlador DMA
Registrador de endereço
Dados
Periférico (disco, pec)

Figura 10.41 Operação de E/S com emprego de DMA.

O controlador DMA coloca um endereço no barramento de endereços para a MP. Ao mesmo tempo ele autoriza, via sinal de controle, que o periférico envie ou receba dados diretamente da memória principal. Quando a operação de transferência de dados é concluída, o DMA retorna o controle do barramento para a UCP.

Um controlador de DMA é um dispositivo bastante complexo. Ele possui diversos registradores internos: pelo menos um registrador de dados, um registrador de endereços e um registrador que armazena um valor igual à quantidade de bytes ou palavras que são transferidas. A Fig. 10.42 mostra uma organização simples de controlador de DMA.

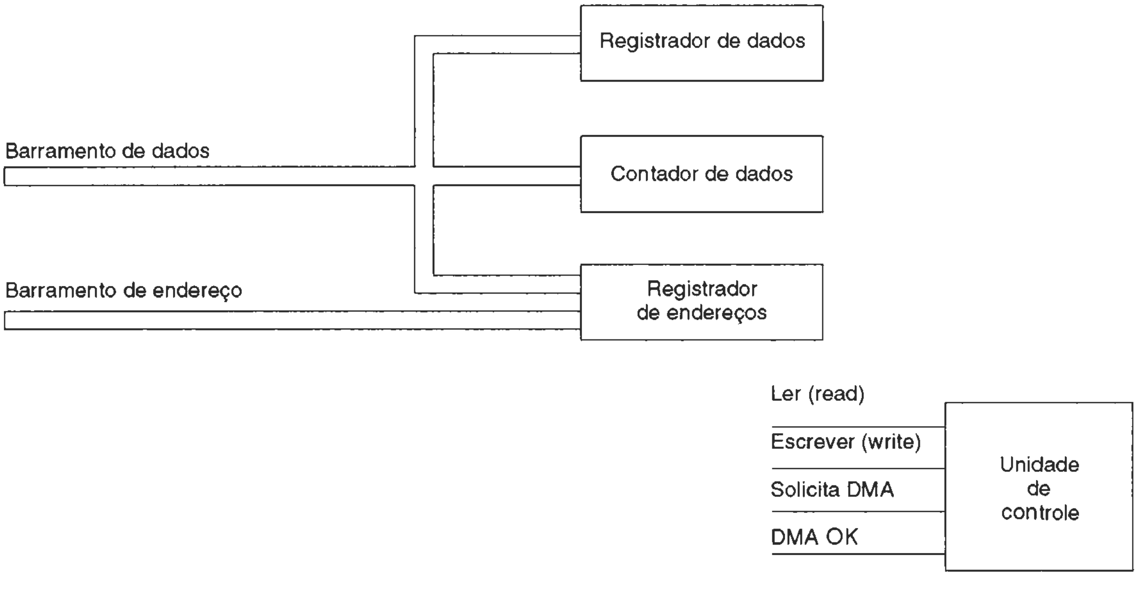

Registrador de dados
Barramento de dados
Contador de dados
Barramento de endereço
Registrador de endereços
Ler (read)
Escrever (write)
Solicita DMA
DMA OK
Unidade de controle

Figura 10.42 Organização simplificada de um controlador de DMA.

10.5 UM POUCO MAIS DE DETALHE

10.5.1 Introdução

Nos itens anteriores foram apresentados conceitos básicos sobre E/S, bem como algumas características dos principais periféricos. Acreditamos que, para os leitores iniciantes e para atender ao programa de muitos cursos, o assunto ali tratado esteja na medida adequada.

No entanto, há leitores e cursos que requerem mais detalhes sobre alguns dos tópicos abordados. A exemplo de alguns capítulos anteriores, o item "Um Pouco Mais de Detalhe" tem o propósito de esclarecer e atender aos que pretendem obter um pouco mais de informações sobre o sistema de E/S de um computador.

Para tanto, este item foi subdividido nos seguintes tópicos: 10.5.2 — Teclado: descreve-se um esquema de geração dos códigos produzidos pelo pressionamento das teclas, bem como algumas informações adicionais e até mesmo históricas sobre esse tradicional dispositivo; 10.5.3 — Vídeos: são apresentados alguns dados adicionais, bem como informações sobre outras tecnologias utilizadas atualmente na fabricação de vídeos para computadores; 10.5.4 — Impressoras: são fornecidos detalhes sobre outras tecnologias de impressão, especialmente em cores; 10.5.5 — Scanners: são apresentados dados e características desse tipo de dispositivo de E/S e 10.5.6 — CD-ROM: são apresentadas algumas características relativas ao funcionamento básico desses periféricos, incluindo-se os mais recentes lançamentos da indústria.

10.5.2 Teclado

10.5.2.1 Etapas Básicas do Funcionamento de um Teclado Utilizado em Microcomputadores

Conforme descrito no item 10.3.1, o teclado da grande maioria dos sistemas de computação funciona utilizando o princípio da interrupção. Além disso, no caso de microcomputadores, é utilizado um código especial, código de varredura (*scan code*), que é transmitido para o subsistema UCP/MP, em vez do próprio código ASCII correspondente ao símbolo pressionado.

Vamos descrever, de forma mais detalhada, os eventos que ocorrem entre o instante do pressionar de uma tecla pelo digitador e o armazenamento, na área de memória do teclado (buffer do teclado), dos bits que representam o código de armazenamento do símbolo (em geral é o código ASCII).

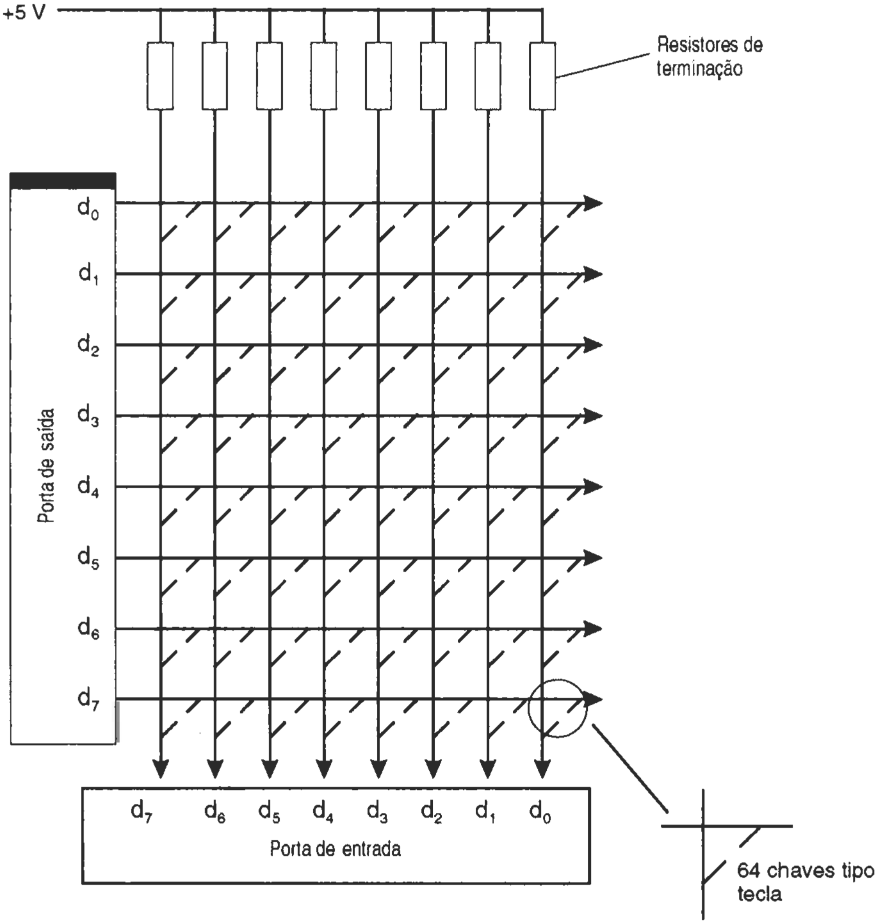


Figura 10.43 Organização de um sistema de codificação de teclas de um teclado.

A Fig. 10.43 mostra a organização de um teclado, constituindo-se em uma matriz de linhas e colunas que cruzam as teclas e identificam, no cruzamento, uma única tecla (a que foi pressionada). O processo de geração do código de varredura consiste na realização de três etapas:

a) detecção do pressionamento da tecla;
b) realização do *debouncing* da tecla; e
c) codificação (produção do código de varredura correspondente).

A primeira tarefa requer um contínuo monitoramento dos circuitos do teclado para determinar (é o único meio) quando uma tecla foi pressionada e, em seguida, identificá-la. Este loop de interrogação é, nos teclados atuais, realizado por hardware, por um microprocessador específico e dedicado, de modo a desobrigar a UCP dessa tarefa. Em microcomputação, é comum o emprego do microprocessador Intel 8048 ou do controlador de teclado e vídeo Intel 8279 para executar as etapas relacionadas anteriormente.

As linhas da matriz são conectadas ao dispositivo de saída (portas de saída) e as colunas são ligadas às portas de entrada (ver Fig. 10.43). Quando não há qualquer tecla pressionada, as colunas são mantidas com valor alto (bit 1) devido aos resistores existentes na sua terminação para a fonte de +5 V.

Quando se pressiona uma tecla, imediatamente o sistema de controle detecta a ligação que ocorre entre a linha e a coluna correspondente (o modo de detecção depende do tipo de tecla). Por exemplo, em sistemas de microcomputadores, que utilizam o processador Intel 8048, o teclado é constituído de teclas capacitivas (ver item 10.3.1) e há, no esquema, um amplificador para "sensar" a diferença de capacitância que é gerada quando uma tecla é pressionada.

Se em uma linha está uma tensão baixa (bit 0) e uma tecla daquela linha é pressionada, então a tensão baixa aparece na coluna que também contém aquela tecla e isto pode ser detectado na porta de entrada (porque linha e coluna foram conectadas pela tecla pressionada). Se houver um meio de identificar qual linha e qual coluna correspondem à tecla pressionada, esta poderá ser identificada e será, então, gerado o código de varredura correspondente.

A Fig. 10.44 mostra um fluxograma do procedimento para detectar o pressionar de uma tecla, confirmar a detecção (*debouncing*) e produzir o código hexadecimal correspondente. Este fluxograma é, como já mencionado, geralmente implementado por hardware/software dedicado (uma placa de circuito impresso, localizada na parte inferior do teclado, contém os elementos necessários à realização de todo o processo de transmissão do código de varredura para a memória principal).

O primeiro passo é zerar todas as linhas. Em seguida, as colunas são lidas, uma a uma, até que todas estejam com valor alto (bit 1). Isto é realizado para se assegurar de que não há qualquer tecla pressionada (a tecla anteriormente pressionada pode ainda não ter sido liberada, isto é, o digitador ainda está pressionando).

Assim que termina este loop de verificação do valor alto das colunas, o sistema entra em outro loop, no qual ele continuamente verifica se já apareceu um valor baixo em uma das colunas (ele varre uma por uma), o que indicará quando uma tecla foi pressionada. Para confirmar que realmente houve pressionamento da tecla (e não um erro qualquer), o sistema aguarda um tempo (em geral é da ordem de 20 ms), o que caracteriza o evento de confirmação (*debouncing*). Após o tempo de espera, é realizada outra verificação do valor da coluna: se todas estiverem com valor alto é porque o valor anterior (baixo) foi acarretado por um ruído e não pelo pressionar de uma tecla. Porém, se ainda permanecer o valor baixo naquela coluna, então é porque realmente a tecla foi pressionada (este procedimento pode ser repetido uma ou mais vezes para garantia).

O passo final consiste em identificar qual linha e qual coluna correspondem à tecla pressionada e em converter a informação obtida no código de varredura correspondente à tecla pressionada. Para isso, é colocada tensão baixa em uma linha e o valor de cada coluna é lido. Se nenhuma das colunas estiver com valor baixo, é porque a tecla pressionada não se encontra naquela linha, sendo o processo então repetido para a linha seguinte e, sucessivamente, até que seja encontrado um valor baixo em uma coluna.

Nesse instante são lidos o código binário correspondente à identificação da linha (no exemplo da Fig. 10.43, correspondente a um teclado 8 × 8, o valor binário teria 8 bits) e o código binário correspondente à coluna detectada. Ambos os códigos servem de entrada em uma tabela localizada na ROM existente, de onde se obtém como resultado o código de varredura correspondente, o qual é transmitido para a memória principal. A transmissão desse valor é usualmente serial, mas há sistemas que utilizam transmissão paralela.

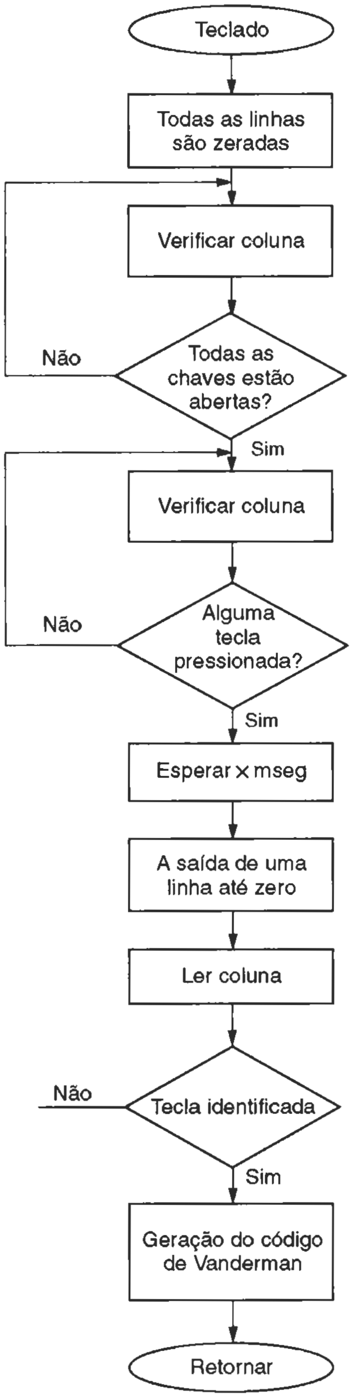

Teclado
Todas as linhas são zeradas
Verificar coluna
Não
Todas as chaves estão abertas?
Sim
Verificar coluna
Não
Alguma tecla pressionada?
Sim
Esperar x mseg
A saída de uma linha até zero
Ler coluna
Não
Tecla identificada
Sim
Geração do código de Vanderman
Retornar

Figura 10.44 Fluxograma do processo de detecção do pressionamento de um tecla, *debouncing* e codificação, em um teclado.

Uma última observação pode ser feita no que se refere aos teclados utilizados em microcomputadores. Nesses dispositivos, cada tecla possui dois códigos de varredura, um correspondente ao pressionamento da tecla e outro correspondente à liberação da tecla pelo digitador. Isto proporciona maior flexibilidade aos programas aplicativos, que podem programar ações para serem desencadeadas com o pressionar de uma tecla e outras para quando a tecla é liberada.

10.5.2.2 Um Pouco de História

A organização das teclas nos teclados usados na maioria dos computadores segue um padrão denominado QWERTY, assim denominado devido à ordem das seis primeiras letras constantes da primeira fileira da parte superior do teclado a partir da esquerda.

Este padrão se mantém o mesmo desde as primeiras máquinas de escrever, criadas no século passado, mais precisamente na década de 1860, cujo inventor foi um tipógrafo americano, Christopher Latham Sholes.

Muitas pessoas devem perguntar por que utilizar uma organização de teclas sem qualquer ordem mais intuitiva ou lógica, como, por exemplo, a ordem alfabética, nossa conhecida. Mas essa ordem (alfabética) foi justamente a escolhida inicialmente por Sholes para sua máquina. Ela continha duas fileiras de teclas, organizadas em ordem alfabética. As letras vinham em alto relevo na ponta de uma barra presa a cada tecla. No entanto, em virtude dos inúmeros choques e enganchamento de duas teclas contíguas quando pressionadas em seqüência, levaram o tipógrafo a tentar diferentes arranjos de teclas que evitassem ou, pelo menos, minimizassem o tal problema de enganchamento das teclas.

A organização atual das teclas surgiu, então, depois de várias tentativas de posicionamento das teclas. Sholes procurou encontrar um posicionamento que afastasse entre si teclas mais utilizadas na língua inglesa, como T e H, por exemplo.

A primeira máquina de escrever (o nome em inglês foi patenteado como Type Writer) comercial foi lançada pela empresa Remington, que havia se associado com Sholes e usava a organização padrão QWERTY, que permanece até hoje.

Como era de se esperar, a referida máquina e sua esquisita organização de teclas teve muitos críticos ao longo do tempo, bem como sofreu várias tentativas de substituição por parte de desenvolvimentos concorrentes. Mas somente muito mais tarde, na década de 1930 é que surgiu uma alternativa que parecia mais eficiente. Essa organização, desenvolvida por August Dvorak (não tem qualquer relação com o conhecido jornalista de informática John Dvorak), um professor da Universidade de Washington, colocava numa mesma linha as principais letras do alfabeto, isto é, as que tinham uso mais freqüente, AOEUIDHTNS, o que, segundo ele, acelerava a produtividade dos datilógrafos, pois o datilógrafo utiliza em grande parte do seu tempo letras de uma mesma linha. A Fig. 10.45 mostra o teclado com a organização Dvorak.

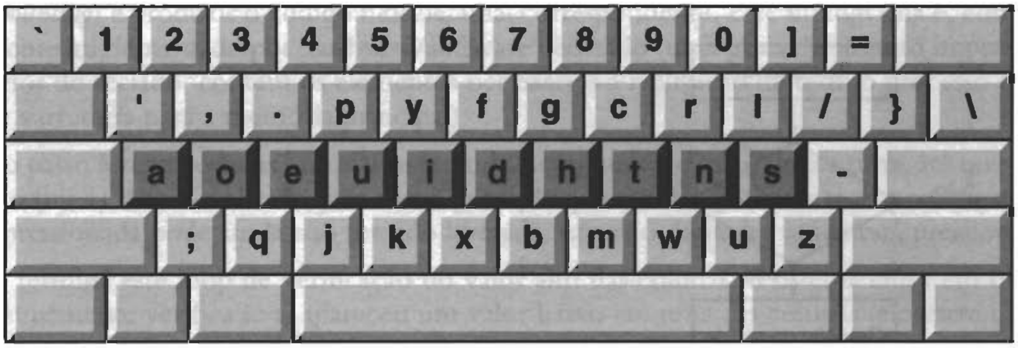


Figura 10.45 Teclado Dvorak.

No entanto, apesar de algumas comprovações sobre a maior eficência do teclado Dvorak sobre o QWERTY, este permaneceu com muito maior preferência pelo mercado, talvez devido à força dos fabricantes que não desejavam mudar algo que estava funcionando bem e dos datilógrafos que já tinham aprendido e se adaptado ao método QWERTY.

10.5.3 Outras Informações sobre Vídeos

Como complemento das informações já apresentadas sobre o funcionamento dos equipamentos de vídeo para computadores, no sentido de esclarecer melhor alguns dos pontos abordados no item 10.3.2, temos:

1) Como já vimos antes, os pixels são cada um dos milhares de pontos formados na tela do vídeo e que servem para constituir uma determinada imagem, um caractere, uma linha ou outro símbolo qualquer. Em um vídeo colorido, um pixel é constituído de três pontos próximos, uma tríade de três cores: vermelho, verde e azul.

Na realidade, a tela do monitor é uma teia composta de milhares de "buracos", que em inglês denomina-se *dot* (no caso de vídeos coloridos cada "buraco" é uma tríade de três "buracos" menores), cada um deles permitindo a passagem do feixe de elétrons, que, acendendo, produz o ponto luminoso na tela. Esta "máscara" sobre a tela facilita a qualidade da imagem por impedir que um ponto ao lado do que foi selecionado se acendesse indevidamente. *Dot pitch* é a distância entre esses "buracos", ou seja, o intervalo entre os *dots*.

2) Durante o funcionamento do sistema, o interface de vídeo é o responsável pela geração do feixe de elétrons e seu acendimento ou apagamento na tela. Deste modo, a quantidade de pontos possíveis de serem gerados nas linhas e colunas (resolução) é função das freqüências horizontal e vertical utilizada pelo interface. No entanto, é importante mencionar que não é só isso que permite termos uma resolução de 800 × 600 ou ainda 1024 × 768, pois é preciso termos suficientes "buracos" na máscara existente na tela. Assim, a resolução adequada é um fator dependente não só do interface como também do próprio monitor.

3) A forma visual, isto é, a aparência com que cada pixel aparece na tela do vídeo colorido, depende da intensidade do feixe colorido correspondente. Se os três feixes forem acionados com máxima intensidade, a combinação das cores gerará o branco, ao passo que, se ocorrer o contrário, eles tiverem intensidade zero, aparecerá o preto. Valores intermediários de intensidade dos três feixes geram as cores intermediárias.

O valor da intensidade de acendimento dos feixes depende da quantidade de bits usada, resultando em mais ou menos combinações de cores e, portanto, mais ou menos qualidade da imagem apresentada. Naturalmente, se usarmos mais bits por feixe, para obtermos mais qualidade, também gastaremos mais memória, visto que esses bits para cada feixe serão armazenados, é claro.

Valores típicos são:

4 bits — resultando em 16 possibilidades de cores — padrão conhecido: VGA.
8 bits — resultando em 256 combinações de cores — denominado em inglês "256 color mode".
16 bits — resultando em 64K combinações de cores — denominado em inglês "High color mode".
24 bits — resultando em 16M combinações de cores — denominado em inglês "True color mode".

Os bits usados são distribuídos pelos três feixes.

Além da tecnologia de emprego de VRC e varredura de rastro que descrevemos no item 10.12, outras tecnologias de fabricação de dispositivos de vídeo para computadores vêm surgindo e crescendo com o passar do tempo, especialmente com o advento de microcomputadores portáteis, como os laptops, notebooks e palmtops. Entre essas tecnologias, vamos apresentar alguns aspectos mais importantes sobre:

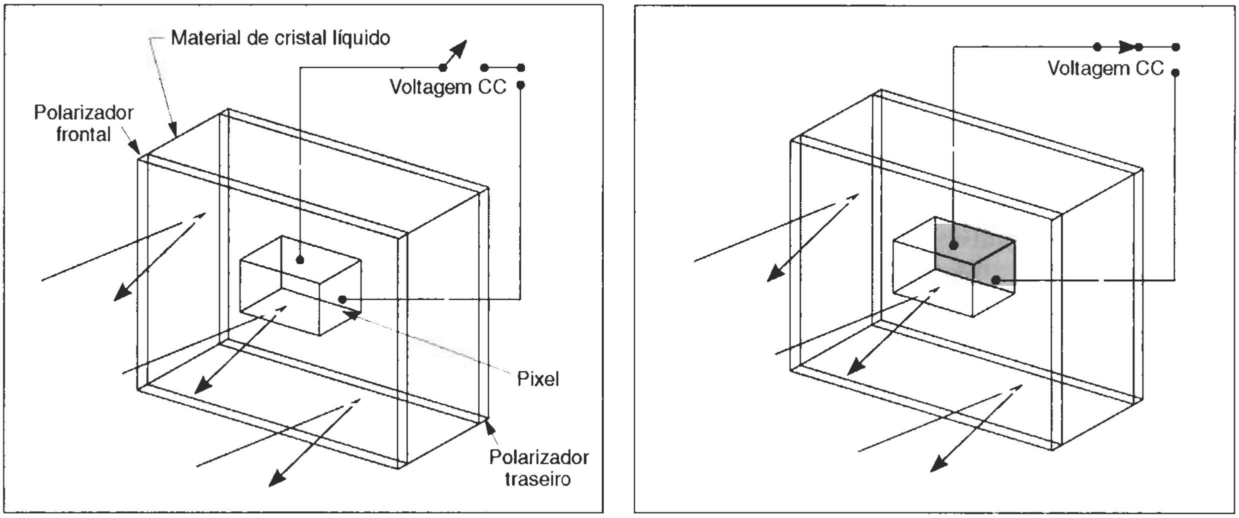
- vídeos de cristal líquido (LCD — Liquid Crystal Display); e
- vídeos de gás plasma.

10.5.3.1 Vídeos de Cristal Líquido — LCD

Vídeos de cristal líquido vêm sendo bastante utilizados em equipamentos portáteis devido às suas vantagens de tamanho, peso e qualidade. O fato de não ser preciso usar um tubo grande e longo como as VRC já é, por si só, bastante atraente. Além disso, os LCD são de pequeno tamanho e peso, possuem baixo consumo de energia e boa resolução (alguns tipos), quase sempre de 800 × 600 pixels e mais. A Fig. 10.46 mostra o esquema de funcionamento de um LCD.

A Fig. 10.46(a) mostra um diagrama simplificado de um típico vídeo monocromático de cristal, denominado *nematic-crystal*. Esse tipo de cristal se situa em um estado intermediário entre o líquido e o sólido e é sensível a campos elétricos. O painel do exemplo consiste em uma matriz de pixels de cristal que podem ser endereçados por um esquema de linha e coluna. Os cristais são colocados entre dois polarizadores e, em sua condição normal, "off ", modificam a polarização da luz que incide sobre o painel, de modo que a maior parte da luz é refletida de volta.

Quando uma voltagem é aplicada (ver Fig. 10.46(b)), as moléculas do cristal modificam sua forma, alterando a polarização da luz incidente. Quando a luz passa através do cristal, ela incide sobre o polarizador posterior e é absorvida. Dessa forma, o pixel endereçado parece mais escuro, comparado com o resto do painel.



(a) Antes da criação do pixel (b) Ativação do pixel

Figura 10.46 Exemplo de formação de um pixel em uma tela de cristal líquido — LCD.

Os primeiros painéis de vídeo do tipo LCD possuíam graves problemas com baixo contraste e clareza das imagens devido a falhas na absorção da luz no cristal, os quais vêm sendo progressivamente reduzidos ou eliminados. Uma das maneiras de melhorar o contraste é utilizar um feixe de luz traseiro (*backlit*), o que reduz a dependência de luz ambiente.

Além de serem utilizados em computadores portáteis, os LCD também estão surgindo em dispositivos projetores, que, acoplados a um retroprojetor, por exemplo, e a um computador, permitem a visualização, em uma tela grande, da imagem do vídeo do computador.

Atualmente vêm sendo fabricados vídeos com tecnologia LCD do tipo de matriz passiva (mais baratos, mas produzindo imagens de pior qualidade, especialmente as coloridas) e de matriz ativa (mais caros, porém com resolução de imagem e de cor muito boa).

Um dos tipos mais produzidos de vídeos LCD emprega uma tecnologia denominada matriz ativa TFT (Thin Film Transistor), que opera mais ou menos dentro do princípio aqui descrito de forma genérica. Os pixels de um vídeo TFT são compostos de três elementos: uma célula vermelha, uma verde e uma azul. Cada célula é composta de uma estrutura nemática e de um filtro de cor vermelha, verde ou azul. Além disso, um transistor TFT controla cada camada do cristal líquido, podendo exercer esse controle de forma bastante precisa, isto é, controlando a diferença de potencial sobre a camada do cristal.

À medida que a voltagem aumenta, as moléculas do cristal se movem gradualmente da estrutura original para uma estrutura mais uniforme. Isto acarreta a possibilidade de a luz incidente girar de 0 a 90 graus, conforme a intensidade da voltagem aplicada.

Dessa forma, o programa de controle do vídeo LCD pode estabelecer precisamente quanto de luz pode passar pela célula de cada pixel no vídeo, com o ajuste correspondente da voltagem. Com isso, o vídeo pode apresentar diferentes cores, muitas na realidade.

Os vídeos TFT ainda são caros em relação aos vídeos de matriz passiva devido à necessidade elevada de substituição de transistores defeituosos durante o processo de fabricação. Cerca de 30% deles sofrem algum tipo de mau funcionamento e mesmo dos 70% restantes ainda alguns aparecem com algum tipo de problema.

Um outro problema desse tipo de vídeo é o consumo de energia, necessário para alimentar todos os transistores que constituem os pixels. Se imaginarmos um monitor com resolução de 1024 × 768, sendo cada pixel constituído de 3 transistores, então somam 2.359.296 transistores, somente no painel.

10.5.3.2 Vídeos de Gás Plasma

Vídeos de gás plasma funcionam seguindo o princípio de excitar um gás, em geral gás néon, através da aplicação de uma voltagem. Uma matriz de eletrodos, separados pelo gás, permite que um certo ponto de tela (o pixel) possa ser endereçado.

Ao se aplicar uma voltagem de valor adequado no ponto de interseção da matriz que foi endereçado, o gás é excitado, emitindo uma luz laranja-avermelhada (esta cor é uma característica dos vídeos de gás plasma).

Como esses dispositivos produzem luz, eles não necessitam de luz externa (*backlightning*), como acontece com os vídeos LCD, porém utilizam mais energia que estes, o que torna esse processo difícil de ser utilizado em sistemas portáteis, que empregam bateria. Para usá-los com bateria, o fabricante teria que incluir no sistema um circuito conversor para alterar a baixa voltagem da bateria na alta voltagem necessária à criação dos pixels (cerca de 200 V) e isto acarretaria uma série de problemas de consumo de energia, complexidade de fabricação e peso.

Além dos problemas com portáteis, os vídeos de gás plasma ainda têm outros tipos de problemas, como a sua incapacidade (pelo menos até então) de fornecer imagens coloridas completas, como, por exemplo, as VRC, além de seu custo ainda elevado, se comparado com outras tecnologias.

10.5.4 Tecnologias Alternativas para Impressão em Cores

A demanda dos usuários por serviços mais sofisticados que os fornecidos com a tecnologia corrente leva sempre à descoberta de novas tecnologias ou ao aperfeiçoamento das existentes. Isto não é diferente no caso de impressoras. As aplicações impulsionam a demanda por saídas coloridas, e a indústria de software não faz por menos ao lançar produtos voltados para a apresentação e editoração eletrônica, para manipular fotografias e imagens obtidas em equipamentos de *varreduras (scanners),* todos com intensa utilização de cor.

Impressoras coloridas são hoje em dia o padrão da indústria, cada vez mais produzidas e vendidas, utilizando diferentes tecnologias de impressão. A Fig. 10.47 mostra um quadro demonstrativo com as várias tecnologias de impressão em cor atualmente existentes.

Na faixa de dispositivos de preço menor aparecem as impressoras de jato de tinta, tendo suplantado em larga escala as impressoras matriciais devido ao melhor custo/qualidade, até mesmo na impressão de serviços mais sofisticados como os de qualidade fotográfica.

A tecnologia de jato de tinta mais adotada consiste na obtenção de uma gota de tinta somente quando a imagem o requer (chama-se a técnica de gota por demanda — *drop-on-demand*). O líquido é forçado através de pequenos orifícios, utilizando-se para isso um entre dois métodos:

- jato de tinta por calor; e
- piezoeletricidade.

No primeiro método, o mecanismo de impressão usa calor para criar uma bolha, que vaporiza e produz a gota de tinta que vai ser depositada no papel. Impressoras de jato de tinta piezoelétricas utilizam atuadores acionados eletricamente para bombear a tinta de um cartucho.

Um dos grandes problemas com impressoras de jato de tinta, que vêm sendo solucionados com muita pesquisa e novas descobertas, reside na interação entre a tinta e o papel. A tinta, ao ser tornada fluida o suficiente para passar pelos diminutos orifícios da cabeça de impressão, também pode se tornar fluida o suficiente para penetrar nas fibras do papel. Além de ter que controlar este problema (o que vem sendo realizado por novas descobertas de material para as tintas), o fabricante precisa controlar a erosão dos bicos de tinta.

Fabricar impressoras a laser monocromáticas já é uma tarefa amplamente dominada, que tem permitido a diversos fabricantes (as impressoras da Hewlett-Packard, HP LaserJet, são vendidas em larga escala no mundo inteiro, bem como as da IBM e Epson) construir modelos confiáveis, de excelente custo/desempenho e com alta qualidade de imagem. Na verdade, na era dos microcomputadores, as impressoras pessoais com tecnologia laser vêm se destacando cada vez mais nas empresas, grandes e pequenas, e até mesmo entre pessoas físicas, devido ao seu notável custo/desempenho.

No entanto, a tecnologia laser para impressão em cores é uma tarefa extremamente mais complexa, que ainda não está permitindo fabricar tais dispositivos por preço competitivo com o das impressoras de jato de tinta ou mesmo de transferência térmica e de sublimação de tinta (no entanto, seu preço continua diminuindo no mercado e, talvez, em breve, possam ser oferecidas com preços aceitáveis para venda em maior escala).

O elemento-chave em um mecanismo de impressão eletrofotográfico é o cilindro fotossensível, no qual a imagem a ser impressa é antes "escrita". Em sistemas de tecnologia laser em cores, a imagem é "escrita" seqüencialmente, uma cor de cada vez, para em seguida ser transferida para o papel e nele fixada por calor. O processo funciona de modo semelhante ao descrito no item 10.3.3 para as impressoras monocromáticas, exceto que:

- há necessidade de quatro cartuchos de toners, um para preto, outro para amarelo, outro para magenta e, finalmente, um para azul (cyan);
- o cilindro deve estar liberado da imagem anterior, em amarelo, por exemplo, antes de se iniciar a escrita da mesma imagem em magenta, e assim também para a mesma imagem em cyan e em preto;
- antes de a imagem final ser fixada, após os quatro passos, o toner deve ser protegido.

Tipo	Tecnologia	Vantagens	Desvantagens
Matricial	Fitas de impressão com cores distintas	Baixo custo	Baixa qualidade de impressão
Jato de Tinta	Gotas por demanda (térmicas e piezoelétricas)	Boa qualidade de impressão Preço baixo	Lenta Absorção de tinta pelo papel Desvanecimento da imagem
Laser	Cilindro fotossensitivo, mais 4 torres uma para cada cor	Imagem durável Vários tipos de papel Qualidade	Lenta Muito cara
Transferência Térmica de Cera (Thermal-Wax)	A tinta (cera) é transferida para o ponto no papel por calor	Pureza Simples e confiável Elevada saturação de cor	Dependente do tipo de papel Pode ser cara
Sublimação de Tinta (Dye Sublimation)	O corante é transferido para o papel por calor	Melhor qualidade de imagem do mercado	Cara para adquirir em material Lenta

Figura 10.47 Quadro demonstrativo de características de impressoras que trabalham com cores.

Outra tecnologia de impressão em cores existente no mercado denomina-se *transferência térmica de cera (thermal-wax-transfer)* e consiste basicamente em um mecanismo de impressão constituído de cabeças de impressão fixas (a quantidade destes elementos por polegada indica a *resolução* da impressora) que contêm dispositivos de aquecimento. O sistema usa o calor para mover o colorante de uma fita de impressão para o papel. Os circuitos de controle da impressora e os programas acionam adequadamente as cabeças de impressão nos locais do papel onde se deseja a tinta para formar a imagem.

Na maioria dos sistemas desse tipo, a fita de impressão é constituída de faixas de cores seqüenciais, do tamanho do papel, ou na ordem azul (cyan), magenta e amarelo ou azul, magenta, amarelo e preto (que garante mais qualidade). A tinta e o papel percorrem juntos o percurso sob a cabeça de impressão. Após um painel de uma cor ser transferido para o papel, este é movido para trás, de modo que outra cor possa ser acrescentada ao papel. Uma vez que a tinta e o papel estão em contato um com o outro quando o calor é aplicado ao conjunto, o colorante tipo cera se desprende (pelo calor) da fita e é transferido para o papel.

A tecnologia de *sublimação de tinta* (*dye sublimation*), também chamada de *transferência térmica por difusão de tinta* ou impressora de tom contínuo, é característica de uma classe extra de impressora, que produz imagens quase tão boas quanto as que vemos em fotografias coloridas. Estas máquinas, embora ainda lentas, continuam a ser a melhor opção do mercado se se deseja imagem em tom contínuo, padrão fotografia. Com esta apreciável vantagem, as impressoras de *sublimação de tinta* também têm muitas desvantagens, a primeira delas diz respeito à baixa velocidade de impressão (a mais rápida imprime uma página por minuto), vindo depois o custo, principalmente de material, como o papel.

O mecanismo de impressão é similar ao das impressoras de transferência térmica de cera, com algumas variações importantes. A unidade pode variar o calor quando está transferindo o corante, o que permite até 256 passos para cada pixel, com os corantes misturados para formar milhões de cores. O resultado é uma imagem em tom contínuo, próxima da qualidade de uma fotografia.

Quando o calor é aplicado na fita de impressão, o corante vaporiza-se, produzindo cores mais densas (ver parágrafo anterior) à medida que mais calor é aplicado.

10.5.5 Scanners

Os *scanners* (poderíamos traduzir como dispositivos de varredura, porém o mercado nacional continua usando o nome em inglês, razão por que o mantivemos neste texto) são dispositivos de E/S (na realidade, são apenas de entrada), que convertem uma imagem existente em um tipo de papel (nem sempre pode ser em qualquer papel) em pontos, os quais são codificados em forma binária. Assim como o mecanismo de criação de imagens em um vídeo é também por pontos, o scanner decompõe a imagem em pontos e os armazena na memória para posterior reprodução ou manipulação.

Um scanner pode ser a "visão" de um computador, pois, contendo milhares de células que funcionam de modo semelhante à visão composta de uma abelha, ele pode "ver" fotografias, textos e imagens. Um *scanner* funciona de modo parecido com o de uma copiadora, bastando colocar uma folha de papel na superfície copiadora do scanner (de vidro) e iniciar o programa de controle e, logo depois, uma cópia da imagem no papel é transportada para a memória do sistema.

A Fig. 10.48 mostra um esquema do processo de funcionamento de um scanner, cujo mecanismo de varredura é formado basicamente de um gerador de luz (em geral é o elemento móvel, uma barra com um emissor de luz), espelhos, lente e um dispositivo produtor dos pontos constitutivos da imagem (CCD — Charged Coupled Device), composto de mais de 2500 elementos ou células fotossensitivas, sendo bem pequeno (cerca de 2,5 cm^2).

O processo se inicia (por controle de um programa) acendendo o gerador do feixe luminoso que percorre o papel, do início ao fim, e a luz, incidindo sobre o papel, é refletida por espelhos e atinge o CCD, produzindo um sinal elétrico em cada uma de suas células. Cada sinal elétrico é proporcional à intensidade da luz refletida que atinge a respectiva célula. Este sinal, que irá constituir-se em um pixel da imagem, é convertido em um código binário e enviado para o computador, de valor menor para os pontos mais escuros e de valor maior para os pontos mais claros.

Os primeiros scanners somente produziam imagens em preto-e-branco, em que cada pixel possuía um número identificado de 1 bit (bit 1, ligado, indicando preto, e bit 0, desligado, para o branco). Atualmente, até mesmo scanners que só funcionam com imagens monocromáticas conseguem distinguir 256 tonalidades de cinza (ou níveis de brilho de luz), quase o dobro do que nossa visão pode distinguir. No entanto, scanners coloridos já são mais vendidos do que os antigos monocromáticos.

A resolução de um scanner é medida em pontos (ou pixels) por polegadas (dpi), valor fixo baseado na quantidade de células do CCD e da área total que pode ser varrida. Por exemplo, um scanner que possui 2590 células e é capaz de efetuar varreduras em uma superfície de 8,5 polegadas tem uma resolução de 300 dpi. Se a área coberta fosse a metade, a resolução seria o dobro.

Os scanners que funcionam com cores capturam a imagem criando três cópias distintas dela, uma para cada cor fundamental da luz refletida pela imagem. Isto é usualmente realizado em três passagens do feixe luminoso pelo papel e, em cada uma delas, um filtro diferente (filtros vermelho, verde e azul funcionam independentemente) registra o componente da luz específica (ou vermelha, ou verde ou azul). Em cada passo, é gerado, como no caso dos scanners preto-e-branco, um código de 8 bits por pixel de cor, produzindo um total de 24

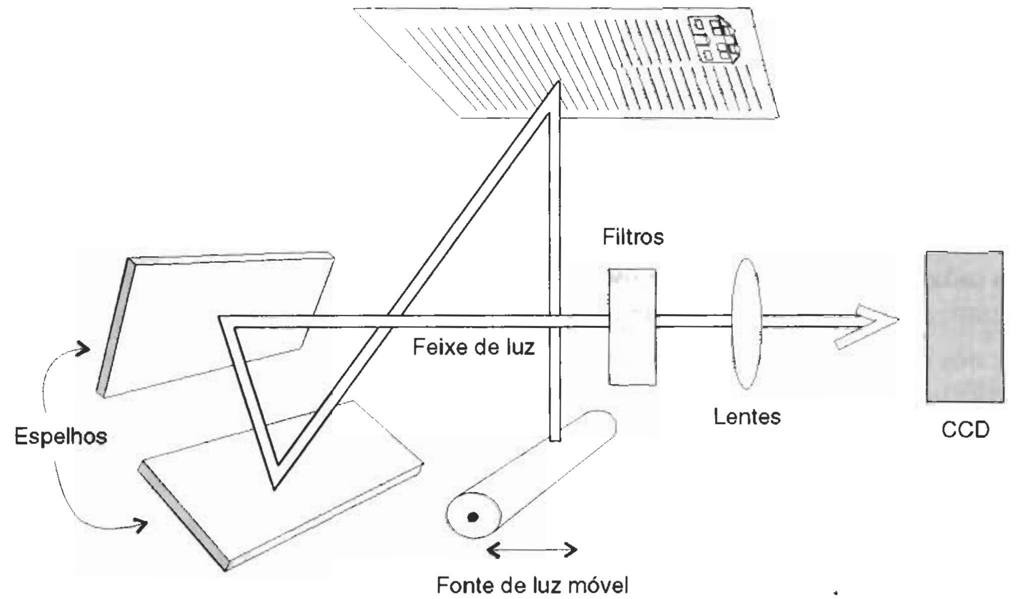


Figura 10.48 Mecanismo de geração de pontos de uma imagem em um scanner.

bits por pixel de imagem. Isto resulta em 16,8 milhões de possíveis cores (2^{24} = 16M), mas também em arquivos bem grandes, demandando sistemas de computação com larga quantidade de memória e velocidade de processamento adequada para processar tantos bits.

10.5.6 Dispositivos de Armazenamento Ótico

Enquanto discos magnéticos armazenam dados sob a forma de variações de polaridade de um campo magnético, discos óticos armazenam bits como variações da reflexão de luz. O disco é constituído de uma superfície circular composta de material altamente reflexivo, organizado em uma única trilha em forma de uma espiral contínua a partir do centro do disco (processo semelhante ao dos antigos discos LP). Estes discos chamam-se *CD* (*compact disks*), como os de áudio, e uma vez que não se pode, por programa, rescrever a informação armazenada neles, foi-lhes dado o nome de *CD-ROM* (*compact disk read only memory*). Os CD-ROM podem armazenar uma grande quantidade de informações, o que é uma vantagem, porém a sua recuperação é bastante lenta (cerca de 250 a 400 ms) se comparada com os tempos de acesso de discos magnéticos (de 12 a 30 ms).

Há algumas razões para que os CD-ROM sejam mais lentos que os discos magnéticos. A primeira delas refere-se à organização de armazenamento. Os CD-ROM armazenam dados em uma trilha em espiral, a qual é ideal para armazenamento de grandes blocos de dados (como em música), mas não para aplicações como leitura e escrita de pequenos registros, devido à dificuldade de localização do registro desejado. Nos discos magnéticos, cuja organização é em trilhas concêntricas, os setores estão sempre em posição fixa e, portanto, são mais rapidamente localizados.

A outra razão para a diferença de velocidade reside no modo como os setores são criados ao longo das trilhas. Nos discos magnéticos, as informações são armazenadas com densidade variável, mas o disco gira com velocidade fixa. A densidade variável (mais bits nas trilhas mais centrais, com menor comprimento, e menos bits nas trilhas mais externas) permite manter a mesma quantidade de bits em todas as trilhas e a velocidade de rotação fixa; no entanto, há uma sobra de espaço nas trilhas mais externas. Já os CD-ROM funcionam de modo diverso.

O disco gira com velocidade diferente, inversamente proporcional ao raio do setor, de modo que todos os setores possuem a mesma densidade. Esta variação de velocidade reduz a taxa de transferência, embora com a vantagem de armazenar o máximo possível de informações no disco (não há espaços mais largos entre bits como acontece nas trilhas mais externas dos discos magnéticos). Esta técnica é utilizada com a finalidade de permitir o máximo de ocupação de espaço e, por isso, obtém grande capacidade de armazenamento, embora se perca em velocidade.

Discos óticos tendem a ser mais vantajosos que discos magnéticos em uma série de itens, porém ainda possuem dois grandes problemas que os têm mantido apenas como elementos de armazenamento secundário aos discos. As duas grandes desvantagens residem na incapacidade de rescrita por programa e na velocidade de transferência, ainda baixa.

Por outro lado, devido à explicação já dada, discos óticos podem armazenar enormes quantidades de dados. Em decorrência da precisão com que um laser pode ser focalizado, as trilhas em um disco ótico são bem mais próximas que as de um disco magnético, além de o espaço necessário para armazenar um bit ótico ser menor do que em meio magnético. Além disso, discos óticos são menos sensíveis à deterioração por poeira ou qualquer outro desses elementos e mesmo por impacto da cabeça de gravação/leitura sobre a superfície (*head crashing*), porque o espaço entre a cabeça e o disco é muito grande (cerca de 1 mm) se comparado com discos magnéticos (algumas cabeças ficam suspensas apenas 0,5 micrômetro).

EXERCÍCIOS

1) O que é um interface de E/S? Como funciona este elemento em um sistema de computação?

2) Descreva o que é tempo de busca (seek) em um acesso de disco magnético.

3) Cite alguns exemplos de interfaces atualmente empregados em computadores.

4) Um disco magnético possui um tempo de busca (seek) médio de 15 ms, um tempo de latência rotacional médio de 8 ms, e ele é constituído de 200 cilindros, cada um com 10 trilhas de 20 setores cada uma. Quanto tempo deverá ser gasto para o sistema ler um arquivo de dados composto de 3000 setores, sabendo-se que o sistema de E/S primeiramente lê todos os setores da trilha 0, começando pelo setor 0, depois todos os setores da trilha 1, e assim por diante, e que o arquivo está armazenado de forma rigorosamente seqüencial?

5) Mostre, através de um desenho, como funciona um vídeo não-entrelaçado. E um vídeo do tipo entrelaçado?

6) Quantas linhas possui um quadro desenhado por um sistema de vídeo do tipo "raster" que opera com freqüência horizontal de 15.750 Hz e freqüência vertical de 60 Hz?

7) Quais são os dois tipos de barramento que podem ser implementados para interligar a UCP aos componentes de E/S?

8) O que caracteriza uma transmissão do tipo serial?

9) Explique em linhas gerais o funcionamento de uma transmissão serial.

10) O que caracteriza uma transmissão paralela?

11) Explique em linhas gerais o funcionamento de uma transmissão paralela.

12) Cite dois microprocessadores muito empregados em controle de um teclado em microcomputadores.

13) Quais são as três etapas básicas do processo de funcionamento de um teclado?

14) Indique três características inerentes a um dispositivo de E/S e a uma UCP que, sendo diferentes, necessitam de um interface.

15) Quais são as ações básicas realizadas por um interface de E/S para efetivar uma comunicação com o periférico ao qual está conectado?

16) Descreva o funcionamento do mecanismo de impressão de uma impressora do tipo matricial.

17) Idem para uma impressora a laser.

18) Como funciona uma impressora de jato de tinta?

19) Deseja-se armazenar em disco magnético um arquivo de dados contendo 30.000 registros lógicos, cada um com 150 bytes e usando um fator de bloco de 15. O disco possui trilhas com 7200 bytes cada uma e cada bloco (registro físico) utiliza 150 bytes para informações de controle. Quantas trilhas seriam necessárias para armazenar todo o arquivo?

20) Suponha que um arquivo A foi armazenado em disco usando-se um fator de bloco igual a 10 e que ocupou 20 cilindros. O disco possui 18 superfícies de armazenamento e trilhas de 13.030 bytes, sendo o tamanho dos blocos igual a 4335 bytes, dos quais 135 são utilizados para informações de controle.

Pergunta-se:

a) Qual é o número de registros lógicos (NRL), o tamanho de cada registro lógico e o número de blocos (NRF)?

b) Se uma cópia do arquivo fosse transferida para fita magnética, qual seria o comprimento (em pés) da fita usada, considerando que o fator de bloco será mantido, que os bytes de controle dos blocos não serão gravados, que cada gap ocupa 0,75 polegada e que será empregada uma unidade de fita que permite selecionar a densidade de 1600 bpi?

21) O que é uma interrupção em um sistema de computação? Qual a importância do emprego de interrupção em operações de E/S?

22) Descreva o método de Acesso Direto à Memória (DMA) para operações de E/S.

23) Qual a vantagem da tecnologia Winchester no processo de fabricação de unidades acionadoras de disquetes (drivers)?

24) Por que é mais vantajoso armazenar um arquivo ocupando seqüencialmente as trilhas de cada cilindro em vez de ocupar seqüencialmente trilhas de cada superfície?

25) Descreva o funcionamento da transmissão assíncrona.

26) Faça o mesmo para a transmissão síncrona.

27) Considere o caso de uma transmissão assíncrona de 200 caracteres, codificados cada um com 8 bits. O sistema emprega paridade do tipo par e pulso Stop com duração de 1 bit e todos os caracteres foram transmitidos em seqüência, sem intervalo entre eles. Calcule a eficiência e o tempo da transmissão supondo que a velocidade de transmissão foi de 2000 bps (bits por segundo).

28) O que é e como funciona uma UART?

29) Por que um disco magnético de grande capacidade deve ser fabricado com um invólucro lacrado e isento de poeira internamente?

30) Descreva a organização de memória para E/S isolada.

31) Descreva a organização de memória compartilhada (*mapped I/O*).

32) Compare as duas organizações de memória para o uso de E/S, indicando vantagens e desvantagens.

33) Qual é a tecnologia de fabricação de teclas mais difundida atualmente para utilização em teclados de computadores? Como funciona basicamente esta tecnologia?

34) Quais são os tipos de tecnologia para fabricação de vídeos de computadores mais comuns no momento?

35) Para que serve e como funciona um mouse?

36) Qual é a diferença de funcionamento entre o modo texto e o modo gráfico em um sistema de vídeo?

37) O que é um pixel?

38) O que caracteriza a resolução de um vídeo de computador? Cite alguns padrões de resolução de vídeos.

39) O que é um *gap* em uma fita magnética? E o que você entende por densidade de uma fita magnética?

40) Qual é a desvantagem do processo de operação de entrada/saída por programa? Cite uma possível solução para esta desvantagem.

41) Qual é a diferença entre uma interrupção interna e uma externa?

42) O que é *debouncing* em um sistema de controle do funcionamento de um teclado?

43) Como funciona um vídeo de cristal líquido? E um vídeo de gás plasma?

44) Cite uma aplicação atual de vídeos de cristal líquido.

45) Descreva o funcionamento básico de um scanner.

46) Calcule a resolução de um scanner que possui 5180 células e é capaz de efetuar varreduras em uma superfície de 8,5 polegadas.

47) Por que um CD-ROM funciona com velocidades mais baixas do que um disco rígido magnético?

48) Qual é a diferença na organização de armazenamento de dados entre um CD-ROM e um disco magnético?

11

Arquiteturas RISC

11.1 INTRODUÇÃO

A evolução acelerada da tecnologia de semicondutores, levando a indústria a criar processadores cada vez mais velozes, propiciou o surgimento de diversos estudos sobre o aperfeiçoamento da arquitetura de computadores, realizados principalmente no início da década de 1980. Tais estudos foram os precursores de um novo tipo de arquitetura, denominada RISC.

RISC é a abreviatura de "Reduced Instruction Set Computer", computador com conjunto reduzido de instruções, e é a identificação de um tipo de arquitetura de UCP (e, conseqüentemente, de todo um sistema de computação) que se contrapõe à arquitetura até então predominante, denominada CISC — Complex Instruction Set Computer, ou computador com conjunto complexo de instruções.

Praticamente todos os microprocessadores lançados no mercado, desde os primeiros processadores de 8 bits — Intel 8080, Motorola 6800, Z-80 — até processadores de 32 bits, como o Intel 80486 e Motorola MC 68040, além de minicomputadores, como a família VAX-11 e AS-400, até os "mainframes", computadores de grande porte, tiveram sua arquitetura especificada do modo característico CISC, isto é, uma grande quantidade de instruções com variedade de modos de endereçamento, poucos registradores de dados na UCP e processamento controlado por microprograma.

No entanto, desde o aparecimento das estações de trabalho SPARC, lançadas no final da década de 1980 pela Sun Microsystems, dos EUA, o conceito de arquitetura RISC vem se tornando de interesse crítico para todo grande fabricante de computadores. Embora a Sun não tenha sido a primeira a vender sistemas RISC, ela foi a primeira a abrir um mercado de rápido crescimento, tornando-se uma grande companhia.

Posteriormente, outros grandes fabricantes entraram no mercado, como a IBM, primeiro com o sistema RTPC, sem sucesso, e depois com o sistema RS/6000 (que teve seu processador inicial multipastilha modificado com o lançamento, em 1992, do processador em uma única pastilha, o Power PC, projeto conjunto da IBM/Motorola e Apple), uma máquina de ótimo desempenho e boas vendas. Além da IBM, a DEC — Digital Equipment Co. (a DEC foi absorvida pela Compaq) entrou no mercado RISC no início de 1992, com o lançamento das estações de trabalho Alpha; a MIPS também é uma companhia que se especializou em estações de trabalho RISC (Silicon Graphics).

Pode-se dizer que o desenvolvimento de arquiteturas RISC teve início por três caminhos próximos, embora conduzindo a alternativas diferentes. São eles:

1) Projeto da IBM, desenvolvido em meados da década de 1970, resultando em uma máquina de baixo desempenho e, portanto, sem sucesso comercial. A IBM somente ganhou mercado nesta área por volta de 1990 com o lançamento da família de processadores RS/6000 e, posteriormente, com a família Power PC, desenvolvida em conjunto com a Motorola e com a Apple.

2) Estudos na Universidade Stanford, Califórnia, por John Hennessy, que redundaram nos processadores da Mips, empresa criada pelo próprio Hennessy.
3) Estudos na Universidade Berkeley, Califórnia, por David Patterson, que redundaram nos processadores desenvolvidos pela Sun.

A tabela da Fig. 11.1 mostra um quadro comparativo de algumas características entre máquinas RISC e CISC.

Vários estudos foram realizados na época visando a verificar o comportamento das instruções de máquina em relação ao desempenho dos sistemas na execução de programas escritos em linguagens de alto nível.

Sistemas	Tipo	Ano	Qtde. inst.	Qtde. reg.	Tamanho inst.
IBM /370-168	CISC	1973	208	16	16-48 bits
Intel 80846	CISC	1989	147	8	1-17 bits
Intel Pentium	CISC	1993	150	8	1-17 bits
Power PC 601	RISC	1993	184	32-I 32-PF	32 bits
Sparc 10	RISC	1987	52	até 528	32 bits
Alpha 21064	RISC	1992	125	32-I 32-PF	32 bits

Figura 11.1 Quadro comparativo de algumas características de processadores CISC e RISC.

Cada linguagem de alto nível possuía alguns comandos poderosos, criados com a finalidade de facilitar a vida dos programadores mais do que facilitar o desempenho da computação, isto é, facilitar o processamento pelo hardware.

Era uma grande vantagem para os programadores desenvolver programas mais complexos, com menor quantidade de esforço, em vista de os comandos disponíveis nas linguagens possuírem capacidade de realizar muitas tarefas, como o comando CASE, por exemplo.

Porém, as instruções de máquina eram primitivas, ou seja, continuavam realizando tarefas simples e diretas, como adicionar dois valores (ADD), desviar para um endereço (JMP), mover um dado de uma célula para outra de memória (MOV).

Ora, se um único comando de alto nível tinha que ser convertido em várias instruções de máquina, isto significava que:

– estava ocorrendo uma separação acentuada entre as operações em linguagem de alto nível e em linguagem de máquina, o que ficou conhecido como um "gap semântico"; e

– devido a este "gap", freqüentemente havia necessidade de escrever complexos compiladores, capazes de realizar corretamente a conversão entre as duas linguagens. Com isso, ele se tornava mais lento, resultando em ineficiência da execução dos programas. Para reduzir este "gap semântico", os arquitetos de processadores imaginaram as seguintes soluções:

a) aumentar a quantidade de instruções de máquina (ver a Fig. 11.1, com a quantidade de instruções dos processadores CISC) e aperfeiçoar algumas delas para atender aos requisitos de processamento de um comando complexo, tornando-as mais complicadas;
b) incluir mais modos de endereçamento no conjunto de instruções; e
c) utilizar mais microprogramação, para realizar certas operações por firmware.

Ao mesmo tempo, vários pesquisadores começaram a examinar o perfil de programas que eles executavam, conforme já mencionamos. Esse trabalho passou a influenciar os projetistas de hardware na realização de modificações nos processadores. A tabela da Fig. 11.2 apresenta o resultado de um dos estudos realizados, sobre comportamento de programas, por um desses pesquisadores, David Patterson, em 1982. Nele mostra-se a percentagem média de diferentes operações comparadas em programas científicos, programas de emprego geral e de editoração [VARH 92]. E mais ainda, mostra-se também a percentagem de tempo gasta na execução dessas operações e a percentagem de memória alocada para as referidas operações.

Pela observação das tabelas verifica-se que, pelo menos para os programas que foram selecionados para a amostragem, somente uma pequena parte de operações básicas foi utilizada. Além disso, loops e chamadas de funções consomem bastante tempo em acessos à memória, uma operação bem mais lenta se comparada com atividades do processador.

Comando	Ocorrência		Peso nas inst. máq.		Peso em ref. à MP	
	Pascal	C	Pascal	C	Pascal	C
Assign	45%	38%	23%	13%	14%	15%
Loop	5%	3%	42%	32%	33%	26%
Call	15%	12%	31%	33%	44%	45%
LF	29%	43%	11%	21%	7%	13%
Goto	–	3%	–	–	–	–
Outros	6%	1%	3%	1%	2%	1%

Figura 11.2 Quadro comparativo da freqüência de ocorrência de certos comandos de linguagens de alto nível na execução de programas.

Outros estudos também foram realizados sobre o mesmo tema, como o do professor Katevenis, da Universidade da Califórnia, em Berkeley, este mais voltado para o comportamento e consumo de recursos de chamadas de funções (CALL). Em todos, as conclusões eram mais ou menos semelhantes, isto é, era necessário aperfeiçoar o hardware para atender à demanda de recursos pelos programas e não havia muita razão em se ter tantas instruções de máquina, se apenas algumas delas eram utilizadas na maioria dos programas. (Muitas instruções significam muitos códigos de operação e, portanto, muitos bits em cada código. Este fato acarreta uma dupla desvantagem: instrução com maior comprimento, devido ao número de bits do C.Op., e mais tempo de interpretação, pois o decodificador gasta mais tempo para produzir uma saída válida entre 2^n possíveis — sendo N a quantidade de bits do C.Op. e tendo N muitos bits.)

Esses resultados conduziram, em alguns centros de pesquisa, especialmente em Berkeley e na IBM, ao desenvolvimento de uma arquitetura de sistema de computação que contemplasse aquelas conclusões, com o propósito de reduzir o "gap semântico" entre as linguagens de programação e de máquina e acelerasse o desempenho das máquinas. E, então, surgiram os primeiros protótipos de máquinas com tecnologia RISC.

11.2 CARACTERÍSTICAS DAS ARQUITETURAS RISC

11.2.1 Menor Quantidade de Instruções e Tamanho Fixo

Talvez a característica mais marcante de um sistema RISC seja a sua tendência de possuir um conjunto de instruções menor que o das máquinas CISC de mesma capacidade. Daí o nome da arquitetura (RISC — computadores em conjunto reduzido de instruções). A família SPARC, da Sun, possui cerca de 50 instruções, enquanto os VAX-11/780 apresentavam até 303 instruções. O Intel 80486 foi lançado com 147 instruções de máquina, e os atuais Pentium possuem mais de 200 instruções.

Com menor quantidade de instruções e com cada uma delas tendo sua execução otimizada, o sistema deve produzir seus resultados com melhor desempenho, mesmo considerando-se que uma menor quantidade de instruções vá conduzir a programas um pouco mais longos.

Além disso, todas as instruções têm o mesmo tamanho em bits, de modo que isso facilita sua busca, visto que ela pode ser realizada em uma única operação e não há necessidade de verificação do seu tamanho para que o CI — Contador de Instruções possa ser corretamente incrementado.

11.2.2 Execução Otimizada de Chamada de Funções

Outra característica importante da arquitetura RISC, que a distingue da arquitetura CISC, refere-se ao modo de realizar chamadas de rotinas e passagem de parâmetros. Os estudos sobre comportamento dos programas revelaram que chamadas de funções (que consomem razoável tempo do processador) requerem usualmente poucos dados, mas consomem, na transferência, demorados acessos à memória em leituras e escritas.

Enquanto em máquinas CISC a chamada de funções conduz a operações de leitura/escrita com a memória para passagem de parâmetros e recuperação de dados, nas máquinas com arquitetura RISC isto ocorre basicamente no processador, utilizando-se para isso mais registradores que as máquinas CISC; os parâmetros e variáveis são manuseados na própria UCP. A possibilidade de colocação de mais registradores na UCP é possível devido à redução dos circuitos necessários à decodificação e execução de instruções (porque há menor quantidade delas).

Com isso, o desempenho total do processador melhora, já que executa mais otimizadamente as chamadas de funções e estas ocorrem em quantidade apreciável na média dos programas.

11.2.3 Menor Quantidade de Modos de Endereçamento

Para facilitar o trabalho dos compiladores, o conjunto de instruções de máquinas CISC tende a ter muitos modos de endereçamento (embora os atuais Pentium apresentem menos modos, os processadores da família VAX-11 tinham 22 modos).

Uma simples instrução de soma pode ser realizada com os operandos localizados de diversos modos: podem-se somar valores que estão armazenados em registradores; outra instrução pode realizar a mesma soma, porém com um operando na memória e outro em um registrador, ou ainda uma outra instrução pode realizar a operação de soma com os dois operandos armazenados na memória.

No caso das máquinas RISC, a busca por soluções mais simples conduziu à criação, de um modo geral, de apenas dois tipos de instruções: LOAD/STORE, para acesso à memória, utilizando somente o modo direto (ver modos de endereçamento no Cap. 8), e demais operações no processador (as operações matemáticas). Esta técnica simplifica consideravelmente o projeto e a implementação das instruções, reduzindo ainda mais os ciclos de relógio necessários à sua realização.

11.2.4 Modo de Execução com *Pipelining*

Talvez a característica mais relevante da arquitetura RISC seja o uso altamente produtivo de *pipelining*, obtido em face do formato simples e único das instruções de máquina.

Conforme observamos no Cap. 6, a técnica *pipeling* funciona mais efetivamente quando as instruções são todas bastante semelhantes, pelo menos no que se refere ao seu formato e complexidade.

Isto é verdade se imaginarmos que os estágios de uma linha de montagem devem consistir em tarefas semelhantes em tempo e forma para que a produtividade seja maior. O ideal seria cada instrução completar um estágio pipeline em 1 ciclo do relógio, embora isso nem sempre seja alcançado.

Por exemplo, uma linha de montagem de fabricação de um determinado objeto consome 1 hora para montar um único objeto e está dividida em 4 estágios. Se cada estágio estiver organizado de modo que suas tarefas para o processo de fabricação de um objeto durem aproximadamente 15 minutos, então o sistema estará altamente produtivo, completando-se um objeto a cada 15 minutos.

No entanto, se as tarefas dos estágios estiverem desbalanceadas e, por exemplo, o estágio 1 durar 20 minutos, o estágio 2 durar 30 minutos e os estágios restantes completarem em 5 minutos cada um, então a produtividade desejada para a linha de montagem será perdida. Um segundo objeto somente será iniciado 20 minutos depois do primeiro, e não mais de 15 em 15 minutos, como no caso anterior.

Conforme já observamos no item 6.6.2, projetar processadores que executam várias instruções quase que totalmente em paralelo é uma técnica bastante eficaz para acelerar o desempenho dos processadores, reduzindo o tempo de execução das instruções para poucos ciclos.

A tabela da Fig. 11.3 apresenta um resumo das características básicas das arquiteturas RISC, que as distinguem das CISC e contribuem para melhor desempenho do hardware em termos gerais.

11.3 MEDIDAS DE DESEMPENHO

Há no mercado alguns métodos de medir e divulgar o desempenho de processadores e sistemas de computação, bem como diversas unidades de medida decorrentes, os quais, em conjunto, podem confundir o observador, em vez de produzir o resultado esperado, isto é, servir de elemento básico de comparação e auxílio à tomada de decisão em algum procedimento de escolha (ver item 2.4).

Uma das unidades de medida mais conhecida e também ambígua é o MIPS (milhões de instruções por segundo). É ambígua porque cada processador executa uma instrução de modo diferente e ainda porque possui instruções diferentes. Como comparar adequadamente medidas de comportamento desigual?

MIPS não é uma boa unidade de medida de comparação entre processadores RISC e CISC porque pode iludir o observador com os resultados, devido ao princípio conceitual de ambas as arquiteturas. Como as máquinas RISC possuem instruções mais simples, tendem a consumir mais instruções de máquina em um programa do que os correspondentes processadores CISC, e isto pode mostrar um total de MIPS superior, conduzindo a uma possível conclusão errônea para os processadores RISC.

Por exemplo, dois processadores, um RISC e outro CISC, executando um mesmo programa podem levar 1 segundo para completá-lo (o mesmo tempo em ambos). Como o programa CISC, por exemplo, gasta 10.000 instruções e na RISC são necessárias 14.000 instruções (40% a mais), a máquina RISC parece que possui uma

Característica	**Considerações**
Menor quantidade de instruções que as máquinas CISC	• Simplifica o processamento de cada instrução e torna este item mais eficaz. • Embora o processador RS/600 possua 184 instruções, ainda assim é bem menos que as 303 instruções dos sistemas VAX-11. Além disso, a maioria das instruções é realizada em 1 ciclo de relógio, o que é considerado o objetivo maior dessa arquitetura.
Execução otimizada de chamada de funções	• As máquinas RISC utilizam os registradores da UCP (em maior quantidade que os processadores CISC) para armazenar parâmetros e variáveis em chamadas de rotinas e funções. Os processadores CISC usam mais a memória para a tarefa.
Menor quantidade de modos de endereçamento	• As instruções de processadores RISC são basicamente do tipo Load/Store, desvio e de operações aritméticas e lógicas, reduzindo com isso seu tamanho. • A grande quantidade de modos de endereçamento das instruções de processadores CISC aumenta o tempo de execução das mesmas.
Utilização em larga escala de *pipelining*	• Um dos fatores principais que permite aos processadores RISC atingir seu objetivo de completar a execução de uma instrução pelo menos a cada ciclo de relógio é o emprego de *pipeling* em larga escala.

Figura 11.3 Características de processadores RISC.

taxa de MIPS maior, o que não indica nada de concreto, já que ambos executaram o mesmo programa no mesmo tempo.

Outra unidade que vem sendo empregada é o MFLOPS (milhões de operações de ponto flutuante por segundo). É uma unidade mais apropriada para medir a velocidade com que cálculos matemáticos são realizados pelo processador, o que interessa mais especificamente a pesquisadores e cientistas envolvidos com programas científicos ou que processam grandes quantidades de números, como, por exemplo, cálculos meteorológicos realizados em supercomputadores.

No entanto, há dois programas diferentes oferecidos no mercado para calcular MFLOPS, os quais produzem resultados razoavelmente diferentes e, por isso, o usuário, ao empregar o método, deve verificar se o mesmo programa de teste está sendo usado em todos os processadores. Um outro ponto, ainda relativo a MFLOPS, refere-se ao fato de que os dois programas de teste disponíveis estão escritos em Fortran (boa linguagem para processamento do tipo científico). Se um processador foi testado com um programa Fortran e utiliza normalmente programas comerciais escritos em Cobol ou em C, os resultados deverão ser diferentes, para menos (o que pode, às vezes, decepcionar o usuário), pois Fortran tem melhor desempenho que aquelas linguagens quando se refere a cálculos matemáticos intensos.

Atualmente vem sendo empregada para medir o desempenho de máquinas RISC a unidade "SPECmark", assim denominada devido à sigla do comitê que a criou (Systems Performance Evaluation Committee), formado em 1989 por representantes de diversos fabricantes.

O teste SPEC é constituído de 10 programas, seis com cálculos matemáticos e quatro mais concernentes a operações com inteiros e caracteres (escritos em C e Fortran). Eles são compilados no compilador do sistema, onde o teste será executado (para que o compilador não seja uma fonte de incorreção nos resultados), e as medidas são produzidas em "SPECmark", que é uma composição dos resultados obtidos em cada teste.

11.4 OBSERVAÇÕES A RESPEITO DE CISC × RISC

Embora haja atualmente um número razoável de adeptos das máquinas que possuem arquitetura RISC, também há, e em grande quantidade, aqueles que relacionam diversas desvantagens desses processadores, advogando em favor da arquitetura CISC.

Vários podem ser os temas para discussão sobre RISC × CISC, um dos quais se refere ao desempenho do processador na execução de um programa. De modo geral, os vendedores e outros pesquisadores tendem a medir desempenho através de programas de teste (*benchmarks*), já discutidos no item anterior. No entanto, verificamos que os referidos programas possuem uma série de complicações na interpretação de seus resultados em função do tipo de ambiente que utilizaram e da natureza dos testes.

Em princípio, os defensores da arquitetura CISC propugnam que instruções mais complexas redundarão em código-objeto menor (as instruções de máquina, sendo mais complexas, se aproximam em definição dos comandos da linguagem de alto nível que está sendo compilada), o que reduz o consumo de memória (menos instruções) com reflexos no custo do sistema.

Isso não é necessariamente correto se considerarmos que uma menor quantidade de instruções nem sempre acarreta menor quantidade de bits (e é a quantidade efetiva de bits que consome menos memória e a menor custo). Se cada instrução CISC possuir mais operandos que as instruções RISC e se cada um de seus operandos ocupar uma boa quantidade de bits na instrução, então poderemos ter um programa CISC maior em bits do que um programa em máquina RISC, apesar de o programa para o processador RISC possuir maior quantidade de instruções.

Por exemplo, um programa escrito para rodar em um processador CISC pode gastar 150 instruções de máquina; cada uma das instruções possui código de operação de 8 bits, podendo ser de um, de dois e três operandos. Cada campo operando ocupa 18 bits e ainda há um campo para outras ações, com 4 bits de tamanho. Em média, as instruções têm um total de 50 bits. Um programa para realizar o mesmo problema, escrito para rodar em um processador RISC, pode ter 220 instruções, que em média ocupam 32 bits.

As instruções são, em sua esmagadora maioria, de dois operandos, porém os operandos são valores em registradores e, por isso, as instruções não consomem muitos bits para endereçar os dois registradores. Como há relativamente poucas instruções, elas têm um campo-código de operação de 6 bits.

O programa para a máquina CISC gastaria 7500 bits, enquanto o programa para a máquina RISC, mesmo possuindo mais 70 instruções que o do processador CISC, consumiria 7040 bits. Trata-se, evidentemente, de um exemplo simples porém elucidativo, porque os valores apresentados estão próximos da realidade das máquinas atuais.

Outro ponto de debate se refere à rapidez da execução de um programa. Os defensores da arquitetura CISC alegam que estas máquinas executam mais rapidamente os programas escritos em linguagem de alto nível devido à pouca quantidade de códigos binários executáveis. No entanto, o tempo que cada instrução leva para ser executada nem sempre conduz à confirmação dessa assertiva.

Máquinas RISC tendem a executar instruções bem mais rápido porque:

a) as instruções possuem C.Op. com menor quantidade de bits (pois o conjunto de instruções é menor) e, portanto, o tempo de decodificação é menor que o das máquinas CISC;

b) as instruções são executadas diretamente pelo hardware e não por um microprograma. Conquanto um processador microprogramado traga mais flexibilidade ao projeto das máquinas, ele também acarreta uma sobrecarga adicional de interpretação de cada instrução. Máquinas RISC não são microprogramadas e, assim, tendem a executar as instruções de modo mais rápido.

Processadores RISC são também otimizados para operações de uma única tarefa devido ao grande número de registradores que possuem e à grande quantidade de estágios de *pipelining*. Nesses casos, a melhor maneira de obter um bom desempenho dos processadores RISC é executar um programa de teste (um *benchmark*), o qual possui exatamente esta característica: um grande número de operações similares, em uma única tarefa. Interessados em processamento científico podem se apoiar mais nesses programas de teste porque o processamento que fazem é similar ao dos programas usuais de teste. Mas o mesmo não se pode dizer de programas comerciais, que utilizam muita E/S (tarefa que, por exemplo, os programas de teste não realizam).

11.5 SISTEMAS RISC COMERCIAIS

Atualmente há no mercado diversos sistemas que utilizam processadores com arquitetura RISC, cujas aplicações estão crescendo com o tempo, principalmente na área de estações de trabalho e servidores de redes locais, embora já haja oferta de máquinas RISC para processamento comercial de aplicações comuns.

Entre os sistemas existentes podemos citar:

SPARC - lançado pela Sun Microsystems, em 1987;
RS/6000 - lançado pela IBM, em 1990;
ALPHA - lançado pela DEC, em 1992.
POWER PC - lançado pela IBM/Motorola/Apple.

A tabela da Fig. 11.4 apresenta uma distribuição percentual do mercado americano de estações de trabalho RISC em 1993, segundo pesquisa realizada pela empresa Computer Intelligence InfoCorp.

Fabricante/Vendedor	Percentagem de máquinas vendidas/alugadas/leased
Sun Microsystems	44,4%
Hewlett-Packard	21,3%
IBM	12,6%
DEC	6,6%
Silicon Graphics	6,5%
Outros	8,6%

Figura 11.4 Distribuição das estações de trabalho no mercado americano em 1993. (Fonte: Computer Intelligence InfoCorp.)

11.5.1 Processadores SPARC

A Fig. 11.5 mostra o diagrama esquemático da arquitetura do processador SPARC (Scalable Processor Architecture). Ela consiste basicamente na UU, que a Sun denominou unidade de inteiro (IU — Integer Unit), na unidade de ponto flutuante (FPU — Floating Point Unit), no co-processador (CP — Coprocessor), em um gerenciador de memória (MmU — Memory Manager Unit) e em uma memória cache, além do barramento interno que interliga os diferentes dispositivos e o barramento do sistema, para ligação com a memória principal externa e os periféricos.

A arquitetura SPARC tem uma particularidade: ela não possui um único desenho. Na realidade, é uma especificação de como a pastilha processa os dados e define o conjunto mínimo de instruções a ser incorporado ao processador, de modo que, tendo acesso a essa especificação, o projetista poderá especificar seu próprio processador SPARC.

Todos os processadores SPARC compartilham um mesmo conjunto de instruções (cerca de 50) e um espaço linear de memória com endereços de 32 bits. Além disso, o processador pode possuir um conjunto total de 512 registradores (embora nenhum deles tenha ainda se utilizado desse total), distribuídos em janelas de 32 registradores de 32 bits cada um, estando disponível apenas uma janela por vez. Essas janelas são usadas para armazenamento de variáveis e parâmetros de funções, quando chamadas, o que lhes garante rapidez no processamento devido à pouca utilização de acesso à memória, uma das características das arquiteturas RISC (ver item 11.2). Somente quando há mais parâmetros que registradores disponíveis é que o processador precisa recorrer à memória para armazenamento desse excesso de parâmetros (e, nesse caso, o desempenho do sistema se reduz).

A arquitetura SPARC é do tipo Load/Store, sendo as operações restantes executadas nos registradores, como qualquer boa arquitetura RISC.

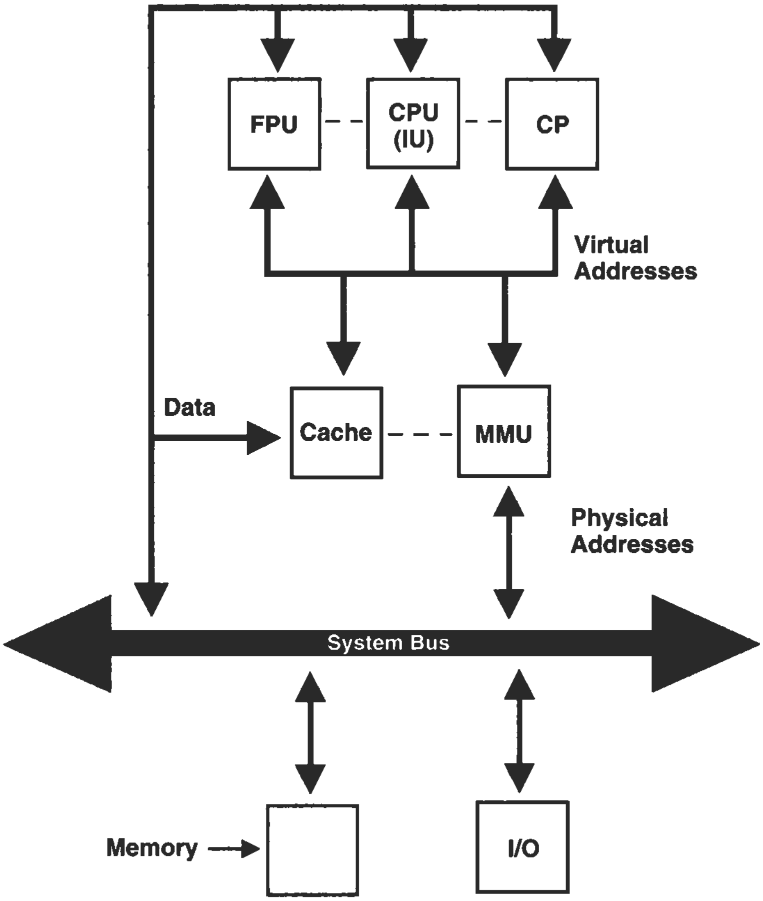


Figura 11.5 Diagrama em bloco da arquitetura SPARC (Sun Microsystems Co.).

O código de operação das instruções possui 6 bits de tamanho (menor que os 8, 16 ou mais dos processadores CISC, o que permite um máximo de 64 instruções (as versões atuais do processador têm cerca de 50 instruções), outra boa característica das arquiteturas RISC.

A maioria dos processadores SPARC é constituída de mais de uma pastilha (uma para a UCP ou unidade de inteiro, outra para a unidade de ponto flutuante, outra para o gerenciador de memória e outra, ainda, para a memória cache), diferentemente dos microprocessadores CISC (inclusive o Intel Pentium), que são compostos de uma única pastilha.

11.5.2 Processadores Power RS/6000

A Fig. 11.6 mostra o esquema lógico do processador RS/6000, o qual é separado funcionalmente em três unidades distintas: o processador de desvios (*branch processor*), o processador de ponto fixo (*fixed-point processor*) e o processador de ponto flutuante (*floating-point processor*). Além dessas três unidades, também são mostradas a memória cache, de dados e de instruções, a memória principal e os dispositivos de E/S. Este processador utiliza atualmente a recente tecnologia desenvolvida em conjunto pela IBM, Motorola e Apple, denominada POWER (Performance Optimization With Enhanced RISC).

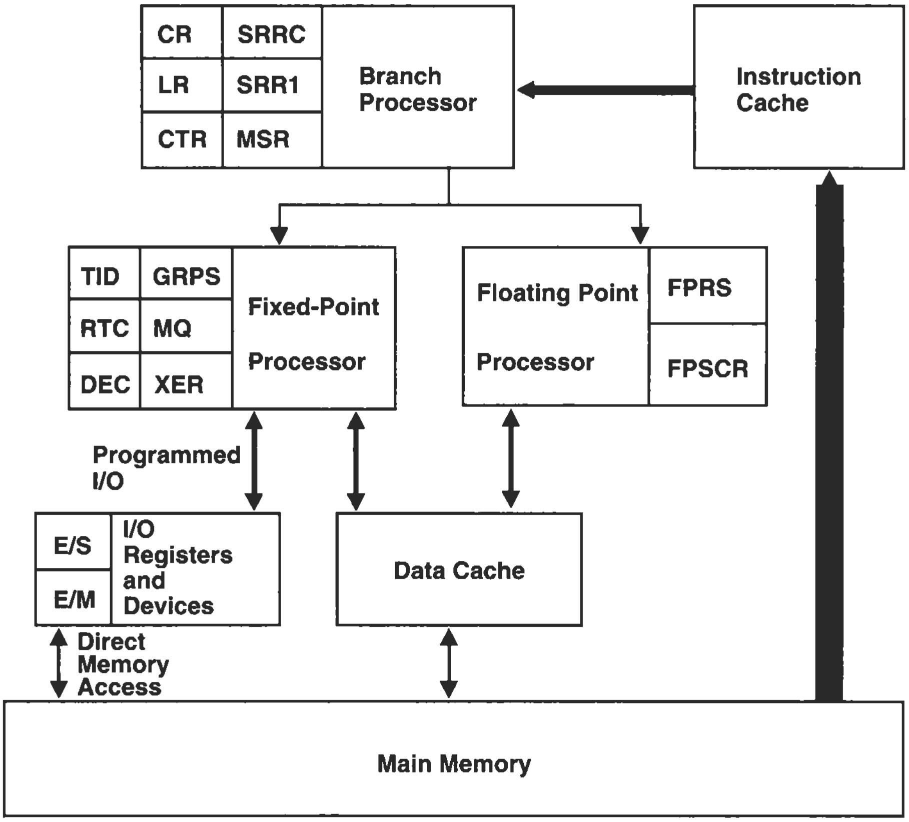

CR
SRRC
LR
SRR1
CTR
MSR
Branch Processor
Instruction Cache
TID
GRPS
RTC
MQ
DEC
XER
Fixed-Point Processor
Floating Point Processor
FPRS
FPSCR
Programmed I/O
E/S
E/M
I/O Registers and Devices
Data Cache
Direct Memory Access
Main Memory

Figura 11.6 Diagrama em bloco da arquitetura do processador RS/6000 (IBM Corp.).

Como resultado da divisão funcional em unidades distintas, o processador RS/6000 pode executar até quatro instruções por ciclo de relógio. Os dados são lidos e escritos entre UCP e MP usando um barramento que pode ter 64 bits ou 128 bits de largura (cerca de 480 Mbytes/s).

A função lógica do processador de desvio é processar o fluxo de instruções recebidas da memória cache de instruções e repassá-las para execução pela unidade de inteiros ou a de ponto flutuante, conforme seja o caso. Ele possui seis registradores especiais.

O processador de inteiros realiza todas as operações aritméticas com valores inteiros (executa as instruções que manipulam aqueles valores), possuindo 32 registradores de emprego geral, cada um com 32 bits de largura, e cinco registradores especiais. Além de processar números inteiros, esta unidade executa instruções que manipulam variáveis lógicas e caracteres.

O processador de ponto flutuante possui 32 registradores de ponto flutuante, de 64 bits cada um e dois registradores especiais, para realizar todas as instruções que manipulam dados representados no formato de ponto flutuante. Cada endereço real nos processadores RS/6000 possui 32 bits de tamanho, permitindo um endereçamento de 4 Gbytes de memória. Os processadores RS/6000, até o advento do Power PC, usavam em seu projeto sete ou nove pastilhas, dependendo de sua configuração (não sendo também um processador de uma só pastilha), contendo mais de 7 milhões de transistores.

Em 1991, foi constituída uma *joint venture* entre a IBM, Apple e Motorola, para desenvolverem um processador que denominaram Power PC. Em 1992, foi lançado o Power PC 601, que passaria a ser o processador das estações de trabalho RS/6000, além de ser implementado em alguns modelos Macintosh, da Apple. O conjunto de registradores é o mesmo em quantidade e tamanho no processador Power PC 601 e no antigo RS/6000, **embora** esteja previsto que um modelo de melhor desempenho, o Power PC 620, seja um completo processador de 64 bits, incluindo os registradores de emprego geral também com 64 bits.

O processador Power PC 601 é composto de uma única pastilha, contendo cerca de 2,8 milhões de transistores, e a quase totalidade de suas instruções é programada diretamente no hardware (e não por microprograma, como nas máquinas CISC).

Em termos de desempenho, os processadores RS/6000 com arquitetura POWER podem alcançar cerca de 75 SPECfp 92 (especificação SPEC para operações em ponto flutuante, versão 1992 — ver item 11.3) e cerca de 63 SPECint 92 (especificação semelhante, porém para operações com inteiros).

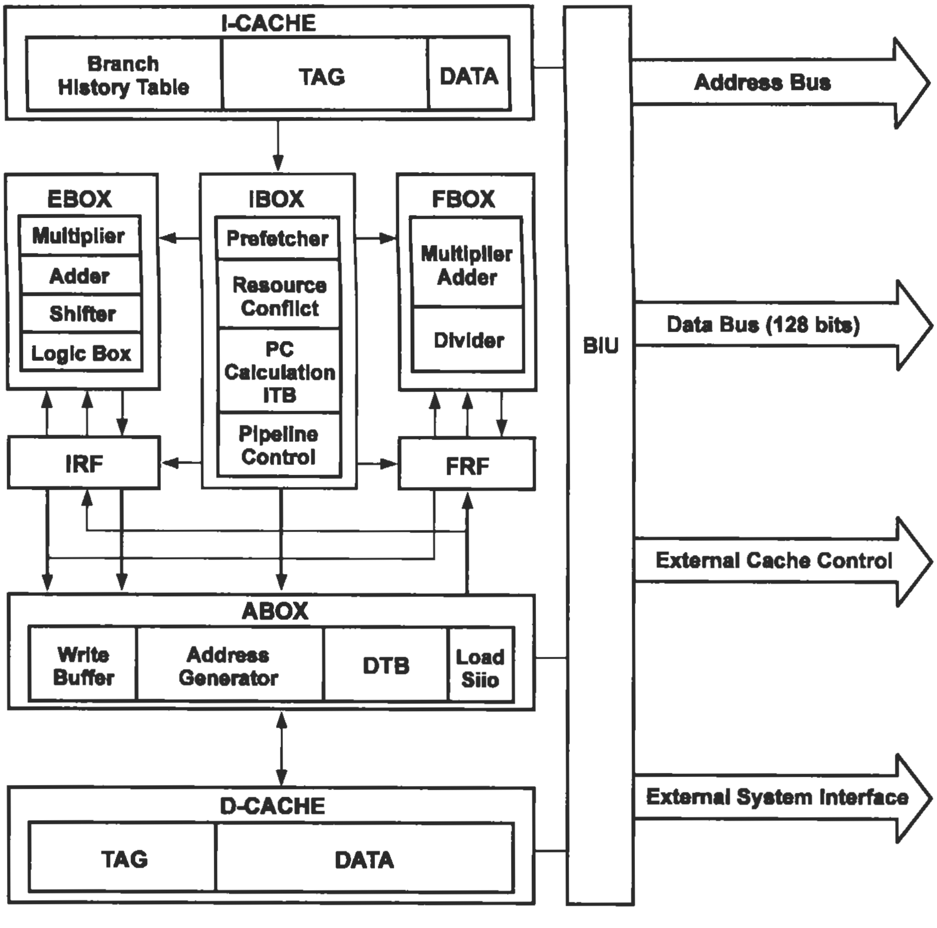

I-CACHE
Branch History Table
TAG
DATA
EBOX
Multiplier
Adder
Shifter
Logic Box
IBOX
Prefetcher
Resource Conflict
PC Calculation ITB
Pipeline Control
FBOX
Multiplier Adder
Divider
IRF
FRF
ABOX
Write Buffer
Address Generator
DTB
Load Siio
D-CACHE
TAG
DATA
BIU
Address Bus
Data Bus (128 bits)
External Cache Control
External System Interface

Figura 11.7 Diagrama em bloco da arquitetura do processador Alpha (Digital Equip. Co.).

11.5.3 Processadores ALPHA

A DEC, atualmente Compaq, lançou seu processador RISC no mercado em fevereiro de 1992, um processador extremamente poderoso e com previsão de elevado desempenho em relação a seus concorrentes. A arquitetura do processador, cujo código de identificação é 21064, é apresentada no diagrama em bloco da Fig. 11.7 e possui, até o momento, as maiores freqüências de relógio do mercado (até 200 MHz, enquanto o processador MIPS R4400 tem relógio de 150 MHz, o processador IBM Power PC 601 funciona até 80 MHz e os processadores SPARC rodam em 50 MHz).

Por esse diagrama podemos observar que o processador possui uma palavra de 64 bits, duas memórias cache distintas, sendo 8 Kbytes para instruções e 8KB para dados, 32 registradores para valores inteiros e 32 registradores para valores em ponto flutuante, uma unidade para operações Load/Store ou de endereçamento (Abox), uma unidade para processamento de inteiros (Ebox), outra para processamento de ponto flutuante (Fbox) e a UCP, denominada lbox, que age naturalmente como uma unidade de controle.

O processador pode executar quatro instruções por ciclo de tempo, sendo duas de valores inteiros e duas com valores em ponto flutuante, e isto é possível graças à cadeia apreciável de estágios de *pipelining.*

EXERCÍCIOS

1) O que caracteriza o chamado "gap semântico", denominação utilizada em estudos sobre processadores na década de 1980?
2) Quais as possíveis providências que podem ser tomadas para reduzir aquele "gap"?
3) Descreva as principais características da arquitetura RISC e compare-as com as arquiteturas CISC.
4) Quais são as principais características do processador Power PC?
5) As arquiteturas RISC se baseiam, entre outros fatos, na existência de um conjunto reduzido de instruções. Como o processador RS/6000 possui 184 instruções de máquina e, ainda assim, registra um excelente desempenho?
6) Qual é a vantagem de processadores possuírem, como acontece em grande parte das máquinas RISC, todas as instruções com tamanho igual?
7) Descreva as principais características da arquitetura SPARC.

Apêndice A

Sistemas de Numeração

A.1 SOBRE SÍMBOLOS E NÚMEROS

Símbolo é uma marca visual ou gráfica que representa um objeto que desejamos identificar, uma idéia ou conceito que desejamos expressar.

A é um símbolo definido para representar a idéia de um caractere, enquanto o símbolo 2 representa o conceito ou idéia de valor (2 litros de leite, 2 reais).

A bandeira nacional é um símbolo representativo do conceito de nação, enquanto *mesa* é um símbolo (constituído de quatro outros símbolos indicadores de cada caractere) que representa um objeto. Para o mesmo objeto, a língua inglesa define outro símbolo, a palavra *table*.

Na aritmética, os símbolos + e − representam, respectivamente, o conceito de adição e de subtração.

Na vida cotidiana costumamos usar indistintamente o símbolo e a idéia que ele representa, como, por exemplo: 2 carros; 2 é o símbolo que representa o valor 2. Embora ocorra essa confusão de usos, devemos ter certeza de que conhecemos a diferença entre o símbolo e seu conceito, especialmente na aritmética, na qual essa confusão é mais freqüente.

Vamos exemplificar o que acabamos de expor:

a) *numeral* é um símbolo designado para representar um número, como, por exemplo: 2, 7, 6 + 9, 57%.

b) *número* é a idéia que o símbolo representa. Um número pode ser representado por diversos numerais, como, por exemplo:

   $5 = 7 - 2 = 4 + 1 = 10/2 = 1 \times 5$

c) considere, nas afirmações a seguir, símbolos representados entre aspas e números (a idéia ou conceito) sem aspas.

Escreva 4, depois × e em seguida 3 e efetue mentalmente a operação ($4 \times 3 = 12$). Representa-se o valor 12 usando os numerais 1 e 2 em seqüência. Em 12, o 1 representa o valor 10 (e não 1) e o 2 representa, realmente, o valor 2.

A origem dos conceitos sobre números não é um fato bem determinado no tempo; sabe-se apenas que o homem pré-histórico já empregava algum modo de contar grandezas, como: um homem, dois peixes ou dois animais.

Os primeiros registros sobre o emprego mais ordenado de números remontam a cerca de 4000 a.C., com as civilizações da Mesopotâmia. Os sumérios e seus sucessores, os babilônios, deixaram muitas informações a respeito do uso de seus sistemas numéricos, relativos a práticas comerciais bastante organizadas.

É importante observar que a simples indicação de nomes para representar números não estabelece a estrutura de um sistema de numeração. Um nome é apenas uma mera representação simbólica de um valor, em uma determinada linguagem; embora o objeto nomeado seja sempre o mesmo, o símbolo usado em cada linguagem costuma ser diferente, conforme já mostrado ao definirmos os símbolos.

Repetindo, com outro exemplo, a grandeza que exprime o valor seis (seis carros, seis pessoas, seis livros, seis reais) é, em português, escrita simbolicamente como seis, enquanto outras culturas possuem nomes (ou símbolos) diferentes para indicar a mesma grandeza, como:

6, VI, six, sechs, IIII II

Na realidade, não há sociedade que tenha criado um nome (numeral) específico para cada número, visto que a quantidade de números é infinita. Em geral, usa-se um método pelo qual são definidos nomes básicos (pequena quantidade); os demais nomes, em seqüência crescente, são estabelecidos através de combinações dos nomes básicos. Trata-se de um método que utiliza a recursividade para compor o nome dos números.

Por exemplo, em nossa linguagem (e em outras também) há nomes específicos para os primeiros vinte e um números (0 a 20), embora alguns desses formem um composto de outros (dezessete é um composto de dez e sete). Em seguida ao vinte, há uma combinação de dois nomes (numerais) entre os já definidos:

vinte e um, vinte e dois, até o vinte e nove, para em seguida aparecer outro nome singular: trinta.

O processo de nomenclatura recursiva prossegue de modo que, a cada nove números (com nomes compostos), é definido um nome novo:

quarenta, cinqüenta, ..., até cem. E depois tem-se mil, etc.

Esse processo de nomear números está relacionado com os símbolos que os representam, visto que podemos criar poucos símbolos diferentes e, com estes, indicar valores crescentes através da comparação desses símbolos, como:

1 2, 1 3 3 3 1 1, X C, C X X I V

A partir desse ponto vamos utilizar o termo *número* em vez de numeral, visto que aprendemos bem a usar a palavra símbolo como se fosse a própria idéia do valor.

A.2 SISTEMA DE NUMERAÇÃO NÃO-POSICIONAL

Atualmente (e desde muito tempo), é generalizado o emprego, tanto na matemática quanto em ambientes comerciais, de um sistema de numeração chamado *posicional*, embora, em tempos bem remotos, alguns povos (os romanos, por exemplo) tenham adotado um outro método representativo de números.

O sistema de numeração romano é constituído de um conjunto N de 7 algarismos diferentes, cada um representando um valor fixo, independentemente de sua posição relativa no número:

N = (I, V, X, L, C, D, M)

indicando, respectivamente, os valores:

1, 5, 10, 50, 100, 500 e 1000.

Nesse sistema, não há símbolo representativo para o zero; os números são definidos da esquerda para a direita, e seus valores obtidos segundo uma regra simples:

- cada algarismo colocado à direita de um maior é adicionado a esse;
- cada algarismo colocado à esquerda de outro maior tem seu valor subtraído do maior.

Exemplo A.1

I V = 4 (I = 1 < V = 5 e à sua esquerda, então: 5 − 1 = 4)

V I = 6 (I = 1 à direita é somado a V = 5)

X C = 90

C L X V = 165 (100 + 50 + 10 + 5)

Observação: É interessante esclarecer que o sistema romano não foi criado para efetuar cálculos matemáticos, devido à enorme dificuldade de efetuá-los em tal sistema.

Exemplo A.2

XXIV	24
+ LXV	65
+ MCXLII	1142
MCCXXXI	1231

Ele era utilizado pelos contadores romanos apenas para registrar informações numéricas (eles usavam ábacos para efetuar os cálculos), sendo baseado no valor 10 e no princípio da adição (posteriormente, criou-se uma complicação com a inclusão da subtração na formação dos números).

Exemplo A.3

$X = 10$; $L = 5 \times 10 = 50$; $C = 10 \times 10 = 100$

$D = 50 \times 10 = 500$ $M = 100 \times 10 = 1000$

$XXVI = 10 + 10 + 5 + 1 = 26$

No entanto, esse sistema possui, conforme já mencionamos, uma notável imperfeição (que tornou impraticável seu uso para cálculos), qual seja, a de inserir regra de subtração na formação dos números (algarismo menor à esquerda de um maior é subtraído desse).

Exemplo A.4

$XIV = 10 + 5 - 1 = 14$, em vez de XIIII

$XL = 50 - 10 = 40$, em vez de XXXX

Ainda a título de ilustração, podemos acrescentar que a forma gráfica dos algarismos romanos evoluiu desde sua criação até se constituir nos algarismos hoje conhecidos. Na realidade, sua invenção data de muitos séculos antes do estabelecimento da civilização romana. Apesar de chamados algarismos romanos, por terem sido amplamente usados por um povo tão importante quanto o romano, sua origem remonta a outros povos,* possuindo antes outra forma gráfica, a qual foi gradualmente evoluindo, conforme se depreende da Fig. A.1.

A.3 SISTEMA DE NUMERAÇÃO POSICIONAL

Um sistema posicional de formação de números é definido pelo fato de o valor de cada algarismo componente do número ser diferente, conforme sua posição no número. Seu valor absoluto é modificado por um fator (ou peso), o qual varia conforme a posição do algarismo, sendo crescente da direita para a esquerda.

Por exemplo, no sistema decimal o número representativo do valor 2622 é constituído de quatro algarismos, tendo três deles o mesmo valor absoluto (o algarismo 2). No entanto, cada um dos citados algarismos indica um valor diferente:

$2622_{10} = 2000 + 600 + 20 + 2$

$2000 = 2 \times 10^3$

---

*Afirma-se que os povos etruscos (século VII ao IV a.C.) foram os inventores desse sistema de numeração, posteriormente chamado de sistema romano.

$600 = 6 \times 10^2$

$20 = 2 \times 10^1$

$2 = 2 \times 10^0$

O fator antes mencionado, que modifica o valor do algarismo conforme sua posição, é, em cada parcela, uma potência de 10 a partir da potência 0 (algarismo mais à direita do número — menos significativo), sendo crescente para a esquerda (potência 1, potência 2, ...).

É uma potência de 10 porque o sistema usado como exemplo é o sistema decimal. Isso conduz a um conceito fundamental dos sistemas posicionais: o de BASE.

Toda a estrutura de formação de números e realização de operações aritméticas em um sistema posicional está relacionada com o valor da base do referido sistema.

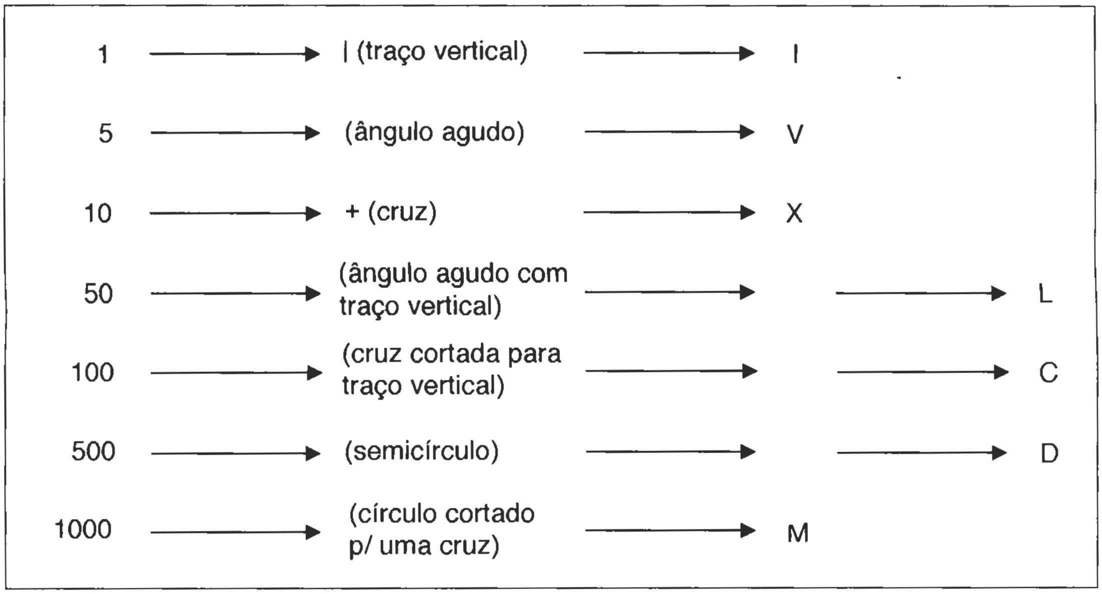

1
| (traço vertical)
I
5
(ângulo agudo)
V
10
+ (cruz)
X
50
(ângulo agudo com traço vertical)
L
100
(cruz cortada para traço vertical)
C
500
(semicírculo)
D
1000
(círculo cortado p/ uma cruz)
M

Figura A.1 Desenvolvimento da forma gráfica dos números.

A.3.1 Base

A noção de *base* de numeração está relacionada à idéia de grupamento de valores, para permitir a contagem e as operações aritméticas de qualquer valor, grande ou pequeno, através do emprego de pequena quantidade de símbolos diferentes.

O problema é originado na necessidade de o homem escrever (ou dizer) números de valor elevado, utilizando, para isso, um mínimo de símbolos possíveis.

Pode-se simplesmente definir a base de um sistema de numeração como a quantidade de símbolos ou dígitos ou algarismos diferentes que o referido sistema emprega para representar números.

O sistema decimal usa 10 símbolos e, portanto, a sua base é 10 (daí o nome decimal).

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

O sistema binário — de base 2 — possui apenas os símbolos: 0 e 1, enquanto o sistema octal — de base 8 — emprega os algarismos:

0, 1, 2, 3, 4, 5, 6, 7

Nos sistemas de base maior que 10, torna-se necessário criar outros símbolos para representar os algarismos não existentes no sistema decimal (algarismos de valor maior que 9). Na base 16 — hexadecimal — convenciona-se usar as letras A, B, C, D, E e F para indicar o valor dos 6 algarismos restantes, que completam o conjunto de algarismos da base 16:

$S_{16} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

O índice 16 indica o valor da base, para distinguir diferentes valores de números com algarismos iguais:

$3\,7_{10} = 3\,7_{8} = 3\,7_{16}$

Adotando para exemplo nosso conhecido sistema decimal, verifica-se que, em vez de criar infinitos símbolos para representar cada número desejado, pode-se grupar valores e simplificar sua representação.

Até o valor 9, os números são escritos com algarismos diferentes, mas o valor seguinte — valor 10 — é representado por dois algarismos — 1 0 —, sendo que o algarismo 1 indica um grupo de 10 unidades (o valor desse grupo é representado pela colocação do algarismo uma casa para a esquerda) e o algarismo zero:

1 grupo de 10 unidades + zero unidades = 10

O número 23, por exemplo, tem seu valor representado por dois grupos de 10 unidades mais três unidades.

Prosseguindo na formação de números de 2 algarismos, atinge-se o maior valor, constituído de nove grupos de 10 unidades mais nove unidades — valor igual a 99. O valor seguinte teria que conter 10 grupos de 10 unidades; não existindo um algarismo 10, avança-se uma casa para a esquerda: uma unidade dessa segunda casa à esquerda representa 10 grupos de 10 unidades ou 100 unidades ou 10^2. E assim por diante.

0
1
2
3
4
5
6
7
8
9
10
11
12
-
-
-
19
20
21
-
-
99
100
101
199
200
201
-
-
999
1000
1001
-
-
-
-

A.3.2 Um Pouco de História

O sistema decimal parece ter sido desenvolvido (ou pelo menos registrado) na Índia, por volta de 600 a.C., tendo sido passado aos persas e transcrito em arábico, razão pela qual foi introduzido na Europa, quando da invasão bárbara, sendo até hoje seus algarismos chamados de *arábicos*.

Os algarismos representativos do sistema evoluíram em sua forma gráfica desde sua origem conhecida até os símbolos atualmente usados, de modo semelhante ao que aconteceu com os algarismos romanos.

O sistema decimal, embora não fosse o primeiro, teve a particularidade interessante da criação do algarismo zero, incluído na constituição dos números. Outros sistemas posicionais mais antigos, como o *sexagenal* (base 60) dos babilônios, e o *duodecimal* (base 12), não possuíam representação para o zero (o sistema não-posicional romano também não possuía representação para o zero).

O sistema de base 12 foi empregado por antigos comerciantes (os sumérios foram um exemplo), sendo bastante interessante devido à sua capacidade de divisão, superior ao sistema decimal; a base 12 é divisível por 2, 3, 4 e 6 (ao contrário do sistema decimal, onde a base 10 é divisível apenas por 2 e 5).

Dessa forma, divisões comuns no comércio (terço, quarto, meio, sexto) são perfeitamente calculáveis e com resultados exatos, no sistema duodecimal. Além disso, há doze meses no ano; 2 vezes 12 horas em um dia; 5 vezes 12 minutos em uma hora, etc.

Atualmente ainda se usa em certas transações comerciais o conceito de uma dúzia, meia dúzia, etc.

Outro sistema de numeração importante na Antigüidade foi o sistema sexagenal (base 60), inicialmente adotado pelos sumérios e depois pelos babilônios, sendo notável o seu estágio de desenvolvimento para a época (cerca de 1800 a.C.). Apesar desse adiantamento, o sistema sexagenal também não possuía representação para o algarismo zero e tinha que recorrer a 60 diferentes algarismos (quantidade grande para se decorar).

Embora fosse difícil efetuar a contagem de números, devido à grande quantidade de algarismos diferentes, a base 60 permaneceu em uso por longo tempo, restando ainda hoje vestígios de sua utilidade na contagem do tempo (hora de 60 minutos e minuto de 60 segundos) e no cálculo de valores angulares (1 grau = 60 minutos e 1 minuto = 60 segundos).

Na verdade, existiram dezenas de modos diferentes de contar valores e efetuar aritmética, todos concernentes a um modo qualquer de representar valores por grupamento de valores menores (conceito de base), cuja identidade depende da posição do algarismo (sistema posicional).

Assim, pode-se definir a estrutura de um sistema de numeração com qualquer valor de base (diferente de 0 e 1), inclusive de bases com valor negativo ou fracionário.

A.4 ALGARISMOS E NÚMEROS

Em um sistema posicional de base fixa B, um número é usualmente representado por uma série de algarismos pertencentes ao conjunto disponível para a referida base.

Assim, dada uma base B, teremos nessa o conjunto S de algarismos:

$S = \{d_{b-1}, d_{b-2}, d_{b-3}, \ldots, d_1, d_0\}$

Cada número que se deseje escrever será representado por:

$$N = d_{n-1}\ d_{n-2}\ d_{n-3}\ \ldots\ d_1\ d_0 \qquad \text{(A.1)}$$

sendo N o número e n a quantidade de algarismos desse número.

Por exemplo, na base decimal (B = 10), temos:

$3\ 7\ 2_{10}$

Sendo n = 3, n - 1 será = 2, na forma de representação mostrada em A.1,

$d_2 = 3 \quad d_1 = 7 \quad d_0 = 2$

Pode-se observar que o conjunto de algarismos exemplificado (372) é interpretado como tendo um determinado valor somente pelo fato de que estabelecemos o conceito de representação posicional.

Desse modo, 372 representa um número cujo valor é obtido ao somarmos três vezes cem, sete vezes dez e duas vezes um. Alterando-se a ordem dos algarismos, obteremos a representação de outro valor, diferente do anterior, como, por exemplo, o valor 237.

$2\ 3\ 7_{10} = 2 * 10^2 + 3 * 10^1 + 7 * 10^0$

Sendo o nosso sistema posicional e decimal (base 10), cada número é interpretado como tendo o seguinte valor:

$$N = d_{n-1} \times 10^{n-1} + d_{n-2} \times 10^{n-2} + \ldots + d_1 \times 10^1 + d_0 \times 10^0 \qquad \text{(A.2)}$$

Observe que nessa outra forma de representação (diferente da apresentada em (A.1)) não há necessidade de indicar-se o valor zero (0), pois o produto de qualquer valor por zero será igual a zero.

No entanto, se o número for representado pela maneira usual (A.1), o algarismo 0 deve constar do conjunto de dígitos.

Por exemplo:

$N = 4 \times 10^2 + 0 \times 10^1 + 6 \times 10^0$

pode ser simplificado para

$$N = 4 \times 10^2 + 6 \times 10^0$$

mas 406_{10} *não é* igual a 46_{10} (sendo necessário o uso do algarismo 0).

A expressão (A.2) pode ser entendida como a soma de fatores com expoentes negativos, indicando valores fracionários. Na realidade, costumamos escrever números mistos sob uma única forma, constituída de uma parte inteira e outra fracionária, separadas por uma vírgula (alguns países usam o ponto em vez da vírgula para separar inteiros de fracionários).

A expressão (A.2) mais completa seria:

$$N = \overbrace{d_{n-1} \times 10^{n-1} + d_{n-2} \times 10^{n-2} \ldots + d_0 \times 10^0}^{\text{parte inteira}} + \overbrace{d_{-1} \times 10^{-1} + d_{-2} \times 10^{-2} + \ldots + d_{-m} \times 10_{-m}}^{\text{parte fracionária}} \qquad (A.3)$$

sendo **n** a quantidade de algarismos inteiros e **m** a de fracionários.

Exemplo A.5

$27{,}3_{10} = 2 \times 10^1 + 7 \times 10^0 + 3 \times 10^{-1} = 20 + 7 + 0{,}3 = 27{,}3$

$32{,}12_{10} = 3 \times 10^1 + 2 \times 10^0 + 1 \times 10^{-1} + 2 \times 10^{-2} = 30 + 2 + 0{,}1 + 0{,}02$

A.5 CONVERSÃO DE BASES

A.5.1 Da Base 10 para uma Base B Qualquer

Um problema comum em computação refere-se à conversão do valor de um número de uma certa base para outra. Isto é, dado um número N, expresso por um conjunto de algarismos de uma base B_o (base origem), procura-se obter o conjunto de algarismos que representa o mesmo número (mesmo valor), expresso em termos de outra base B_r (base resultante).

Sabe-se que o número N pode ser representado por:

$$N = d_{n-1}\, d_{n-2}\, d_{n-3} \ldots d_1\, d_0 \qquad (A.1)$$

ou

$$N = d_{n-1} \times B_r^{n-1} + d_{n-2} \times B_r^{n-2} + \ldots + d_1 \times B_r^1 + d_0 \times B_r^0 \qquad (A.2)$$

onde os algarismos (dígitos) $\mathbf{d_i}$ são os algarismos, ainda desconhecidos, de N, para sua representação na base B_r.

Para se obter os referidos algarismos, pode-se desenvolver a expressão (A.2) sob a forma polinomial multiplicativa, resultando no polinômio mostrado em (A.4).

$$N = \{[(d_{n-1} \times B_r + d_{n-2}) \times B_r + d_{n-3}] \times B_r + \ldots + d_1\} \times B_r + d_0 \qquad (A.4)$$

Por exemplo, poder-se-ia representar um valor na base 10 segundo a expressão (A.2), o que resulta na expressão (A.5):

$$18\,543_{10} = 1 \times 10^4 + 8 \times 10^3 + 5 \times 10^2 + 4 \times 10^1 + 3 \times 10^0 \qquad (A.5)$$

ou segundo a expressão (A.4), o que resulta na expressão (A.6):

$$18\,543_{10} = |\{[(1 \times 10 + 8) \times 10 + 5] \times 10 + 4\} \times 10 + 3| \qquad (A.6)$$

A.5.1.1 Conversão de Números Inteiros

A forma polinomial expressa em (A.4) pode também ser desdobrada de outro modo, se se considerar que:

$$N = N_1 \times B_r + d_0,$$

expressão que denominaremos N_0, e onde N_1 é, na expressão (A.4), todo o polinômio compreendido entre chaves { }.

E os polinômios mais internos são:

$N_1 = N_2 \times B_r + d_1$

$N_2 = N_3 \times B_r + d_2$

$N_{n-1} = d_{n-1}$

Ou ainda:

$N = N_1 \times B_r + d_0$

$N_j = N_{j+1} \times B_r + d_i$

$N_{n-1} = d_{n-1}$

No exemplo apresentado em (A.6) — $18\,543_{10}$ — teremos:

$N = N_0 = 18\,543 = N_1 \times B_r + d_0 = 1854 \times 10 + 3$ ou $d_0 = 3$

$N_1 = 1854 = N_2 \times B_r + d_1 = 185 \times 10 + 4$ ou $d_1 = 4$

$N_2 = 185 = N_3 \times B_r + d_2 = 18 \times 10 + 5$ ou $d_2 = 5$

$N_3 = 18 = N_4 \times B_r + d_3 = 1 \times 10 + 8$ ou $d_3 = 8$

$N_4 = d_{n-1} = 1$ ou $d_4 = 1$

Como, por definição, $0 <= d_i < B_r$, para todos os valores de **i**, podemos obter cada algarismo $\mathbf{d_i}$, o qual é o resto da divisão de N_i por B_r. Ou seja,

$N_{i+1} = [\, N_i / B_r \,]$

As operações de sucessivas divisões são realizadas na base origem (B_o) e, por isso, esse processo é empregado na conversão de valores da base 10 (base origem) para outra base qualquer (B_r).

Exemplo A.6

Converter o número 175_{10} para a base 2 (B_r).

O algoritmo básico consiste nos seguintes passos:

```
a) $i = 0$ (inicialização do contador);
b) valor $N_i = N$;
c) enquanto $N_i$ # 0
    1) $d_i$ = resto da divisão: $N_i / B_r$;
    2) $N_{i+1}$ = quociente de: $N_i / B_r$;
    3) $i = i + 1$.
```

Sendo $N = 175$ e $B_r = 2$, teremos:

i	N_i	N_{i+1}	d_i (resto)
0	175	87	$1 = d_0$
1	87	43	$1 = d_1$
2	43	21	$1 = d_2$
3	21	10	$1 = d_3$
4	10	5	$0 = d_4$
5	5	2	$1 = d_5$
6	2	1	$0 = d_6$
7	1	0	$1 = d_7$

O número, na base 2, é: 10101111_2.

A.5.1.2 Conversão de Números Fracionários

O processo empregado no item anterior aplica-se à conversão de números inteiros, porém o procedimento de obtenção de algarismos fracionários de uma base B_r é bastante semelhante, se considerarmos apenas a parte fracionária do número N, obtida da expressão geral (A.3):

$$N = d_{-1} \times B_r^{-1} + d_{-2} \times B_r^{-2} + \ldots + d_{-m} \times B_r^{-m} \qquad (A.7)$$

onde **m** é a quantidade de algarismos fracionários.

Cada algarismo fracionário do número, para a nova base (B_r), será a parte inteira do produto da base B_r por um valor N_{-i}. O valor N_{-i} é a parte fracionária a ser convertida.

O algoritmo é então:

a) multiplicar B_r pelo valor fracionário a ser convertido;

b) o resultado obtido é um valor constituído de duas partes: parte inteira (mesmo tendo valor igual a zero), e parte fracionária, ambas separadas pela vírgula;

c) a parte inteira compreende o algarismo $\mathbf{d_{-1}}$ desejado (primeiro algarismo à direita da vírgula);

d) a parte fracionária será novamente multiplicada pela base B_r, obtendo-se novo resultado (sempre dividido em duas partes);

e) repetir o processo, a partir do item b) descrito aqui, obtendo-se sucessivamente os algarismos: d_{-2}, d_{-3}, ..., d_{-m}.

A exemplo do que ocorre na conversão de valores inteiros, também nesse caso (valores fracionários) as operações são realizadas com a aritmética da base origem, B_o.

Exemplo A.7

Converter o número $0{,}7265625_{10}$ para a base 2.

-i =	Valor a ser multiplicado por $B_r = 2$	Resultado	Parte inteira = d_{-i}	Parte fracionária: novamente multiplicada
1	0,7265625	1,453125	1	0,453125
2	0,453125	0,90625	0	0,90625
3	0,90625	0,8125	1	0,8125
4	0,8125	0,625	1	0,625
5	0,625	0,25	1	0,25
6	0,25	0,50	0	0,50
7	0,50	1,00	1	0,00

O resultado será $0{,}1011101_2$.

Para verificar a correção do resultado, efetua-se a conversão no sentido inverso:

$$0{,}1011101_2 = 1 \times 2^{-1} + 1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-5} + 1 \times 2^{-7} =$$

$$= 0{,}5 + 0{,}125 + 0{,}0625 + 0{,}03125 + 0{,}0078125 = 0{,}7265625_{10}.$$

Exemplo A.8

Converter o número $0{,}78125_{10}$ para a base 8.

$0{,}78125 \times 8 = 6{,}25$ ou $d_{-1} = 6$ e $N_{-1} = 0{,}25$

$0{,}25\ (N_{-1}) \times 8 = 2{,}00$ ou $d_{-2} = 2$ e $N_{-2} = 0{,}00$

O resultado será: $0{,}\ 62_8$.

A conversão de valores mistos (parte inteira e fracionária, ambas contendo dígitos significativos) deve ser realizada pela execução separada dos dois algoritmos apresentados.

Um fato interessante a considerar é que no caso da conversão de valores inteiros, de B_o para B_r, e no sentido inverso, de B_r para B_o, os resultados apresentam sempre valores exatos, como, por exemplo:

$21_{10} = 10101_2$ e $10101_2 = 21_{10}$.

No entanto, no que se refere à conversão de valores fracionários, as operações de multiplicação da parte fracionária pela base B_r podem ser efetuadas infinitas vezes (em certas conversões) sem que seja possível identificar uma condição exata para terminar o processo (nos exemplos anteriores, o processo de multiplicação terminava quando se obtinha valor 0 para o resultado). Isto ocorre porque não há uma correspondência exata e biunívoca entre cada par de valores reais as bases B_o e B_r.

Nos exemplos anteriores, foram escolhidos números que produziram resultados exatos, mas podemos verificar que, na maioria das conversões, isso não ocorre.

Na prática, é necessário estabelecer a quantidade desejada de algarismos significativos, o que determinará a quantidade de operações de multiplicação a ser efetuada.

É óbvio que, quanto maior a quantidade de algarismos significativos, maior será a precisão do número fracionário. No entanto, quanto mais algarismos ele possuir, maior deverá ser o espaço interno de armazenamento (o Cap. 7 trata de representação de dados e do problema de espaço *versus* precisão).

Exemplo A.9

Converter o número $0{,}37_{10}$ para a base 2.

$0{,}37 \times 2 = 0{,}74$ $d_{-1} = 0$ $N_{-i} = 0{,}74$

$0{,}74 \times 2 = 1{,}48$ $d_{-2} = 1$ $N_{-i} = 0{,}48$

$0{,}48 \times 2 = 0{,}96$ $d_{-3} = 0$ $N_{-i} = 0{,}96$

$0{,}96 \times 2 = 1{,}92$ $d_{-4} = 1$ $N_{-i} = 0{,}92$

Como exemplo, decidimos terminar o processo após quatro multiplicações, obtendo-se como resultado um valor com quatro algarismos significativos:

$0{,}\ 3\ 7_{10} = 0{,}\ 0\ 1\ 0\ 1_2$

Se o valor encontrado for convertido no sentido inverso (base 2 para base 10), teremos:

$0{,}0101_2 = 1 \times 2^{-2} + 1 \times 2^{-4} = 0{,}25 + 0{,}0625 = 0{,}3125_{10}$

Como se pode verificar, ocorreu uma razoável imprecisão entre os valores: 0,37 e 0,3125.

Se, em vez de quatro algarismos significativos, optássemos, na conversão para a base 2, por um valor com oito algarismos fracionários significativos, a precisão seria maior:

$0{,}82 \times 2 = 1{,}64$ $d_{-5} = 1$ $N_{-i} = 0{,}64$

$0{,}64 \times 2 = 1{,}28$ $d_{-6} = 1$ $N_{-i} = 0{,}28$

$0{,}28 \times 2 = 0{,}56$ $d_{-7} = 0$ $N_{-i} = 0{,}56$

$0{,}56 \times 2 = 1{,}12$ $d_{-8} = 1$ $N_{-i} = 0{,}12$

O novo resultado seria:

$0{,}\ 3\ 7_{10} = 0{,}\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1_2$

A conversão no sentido inverso (para a base 10) seria:

$0{,}01011101_2 = 1 \times 2^{-2} + 1 \times 2^{-4} + 1 \times 2^{-5} + 1 \times 2^{-6} + 1 \times 2^{-8} =$

$= 0{,}25 + 0{,}0625 + 0{,}03125 + 0{,}015625 + 0{,}00390625 =$

$= 0{,}36328125_{10}$.

O novo resultado aproxima-se mais do valor inicial 0,37, demonstrando que, quanto maior a quantidade de algarismos, maior será a precisão do resultado.

A.5.2 Conversão de Base B (não 10) para Valor Decimal

O aspecto mais interessante desses procedimentos de conversão (divisões sucessivas) refere-se ao fato de que as divisões devem ser executadas segundo as regras aritméticas da base origem (B_o). Por essa razão, o método é conveniente e simples para a conversão de números da base 10 (B_o) para uma base B_r qualquer, pois, nesse caso, usaremos nossa conhecida aritmética decimal.

Entretanto, também podemos converter números de uma base B_o diferente de 10 para a base decimal (base 10), isto é, efetuar a conversão:

$B_o = 10$ para $B_r = 10$

Nesse caso, como as divisões devem ser realizadas na aritmética de B_o, é preciso que se conheçam previamente os resultados (valores do quociente e resto de cada divisão) a serem obtidos nessa aritmética.

Exemplo A.10

Converter o número 657_8 para representação equivalente na base 10.

O processo consiste, basicamente, nas divisões sucessivas pelo valor 10_{10} (a primeira divisão será do próprio número a ser convertido e, em seguida, as divisões serão dos quocientes a serem obtidos nas divisões anteriores), usando-se aritmética da base 8. Na realidade, a divisão será por $12_8 = 10_{10}$.

Teremos, então:

1) $657_8 \,/\, 12_8$

a) $65_8 \,/\, 12_8 = 58_8$ multiplicado por $B_r = 25_8 \times 12_8 = 62_8$ $65_8 - 62_8 = 3_8$

b) $37_8 \,/\, 12_8 = 3_8$ $3_8 \times 12_8 = 36_8$ $37_8 - 36_8 = 1_8$

Então:

$$\begin{array}{l|l} 657_8 & 12_8 \\ \cline{2-2} \;\;37 & 53_8 \\ \;\;\;\;1 & \end{array}$$

Quociente: 53_8

Resto: 1 ou $d_0 = 1_{10}$

2) $53_8 \,/\, 12_8 = 4_8$ $4_8 \times 12_8 = 50_8$

$53_8 - 50_8 = 3_8$

Então:

$$\begin{array}{l|l} 53_8 & 12_8 \\ \cline{2-2} -50 & 4_8 \\ \cline{1-1} \;\;\;3 & \end{array}$$

Quociente: 4_8

Resto: 3_8 ou $d_1 = 3_{10}$

3) $4_8 \,/\, 12_8 = 0_8$

Quociente: 0_8

Resto: 4 ou $d_2 = 4_{10}$

$$\begin{array}{l|l} 4_8 & 12_8 \\ \cline{2-2} 4 & 0_8 \end{array}$$

O número é: $4\ 3\ 1_{10}$

Ainda com relação ao exemplo apresentado, dois pontos devem ser considerados:

- Para realizar as operações aritméticas em base não-decimal, necessita-se conhecer os resultados das operações na base referida, o que é um processo trabalhoso. Na prática, devem ser estabelecidas previamente tabelas de conversão para os valores que serão manipulados, como exemplificado na Fig. A.2 (exemplo da conversão de base $B_8 \rightarrow$ base B_{10}).
- Tendo em vista o polinômio (A.2), que expressa o valor de um número na base do resultado, com a aritmética dessa base torna-se muito mais simples obter a conversão de uma base B_o qualquer para a base $B_r = 10$, através do referido polinômio (ver item 2.3.2).

Assim, quanto ao exemplo anterior, teríamos:

$657_8 \rightarrow B_{10}$

$6 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 = 431_{10}$

A.5.3 Conversão Direta entre Bases Não-decimais

O método de divisões sucessivas, usando aritmética da base origem, não é apropriado para conversão à base decimal, porém é adequado para conversões diretas entre bases diferentes da base 10, ou seja, onde as bases B_o e B_r são ambas diferentes de 10. Se não houvesse a possibilidade de conversão direta, teria que se efetuar o processo em duas etapas:

1) convertendo da base origem para a base 10;
2) convertendo da base 10 para a base desejada.

Para a conversão direta é necessário apenas que sejam elaboradas tabelas semelhantes à da Fig. A.2, conforme as bases envolvidas, de maneira a se poder efetivar a aritmética na base origem, B_o.

Exemplo A.11

Converter 673_8 para B_6.

B = 10	B = 8	+	0	1	2	3	4	5	6	7
10	12	0	0	1	2	3	4	5	6	7
20	24	1	1	2	3	4	5	6	7	10
30	36	2	2	3	4	5	6	7	10	11
40	50	3	3	4	5	6	7	10	11	12
50	62	4	4	5	6	7	10	11	12	13
60	74	5	5	6	7	10	11	12	13	14
70	106	6	6	7	10	11	12	13	14	15
80	120	7	7	1	11	12	13	14	15	16
90	132									

Figura A.2 Tabela para operações aritméticas em base 8.

a) Dividir 673_8 por 6_8, empregando aritmética de base 8.

b) Prosseguir na divisão, obtendo sucessivos quocientes e restos, que se constituirão nos algarismos de base 6 (B_r).

$673_8 / 6_8$ Quociente: 118_8 Resto: 4; $d_0 = 5_6$

$111_8 / 6_8$ Quociente: 14_8 Resto: 1; $d_1 = 1_6$

$14_8/6_8$ Quociente: 2_8 Resto: 0; $d_2 = 0_6$

$2_8/6_8$ Quociente: 0_8 Resto: 2; $d_3 = 2_6$

O resultado é: 2015_6.

Esse valor pode ser verificado pelo método (mais trabalhoso, porém com menor possibilidade de erros devido à nossa intimidade com a base 10) de conversão intermediária para a base decimal — 10.

Assim, da base 8 para a base 10, teremos:

$673_8 = 6 \times 8^2 + 7 \times 8^1 + 3 \times 8^0 = 443_{10}$

e da base 10 para a base 6:

$$\begin{array}{rrrrrl}
443 & \underline{\vert\ 6\quad} & & & & \\
23 & 73 & \underline{\vert\ 6\quad} & & & \\
5 & 13 & 12 & \underline{\vert\ 6\quad} & & \\
1 & 0 & & 2 & \underline{\vert\ 6\quad} & \\
 & & & 2 & 0 & = 2\,0\,1\,5_6
\end{array}$$

A.6 OUTROS MÉTODOS DE CONVERSÃO DE BASES

A forma polinomial multiplicativa (A.4), que deu origem à dedução do método de conversão por divisões sucessivas, permite também que se obtenha o valor de um número em $B_r = 10$ através do cálculo do polinômio (conforme exemplificado antes, com o valor 18543_{10}).

Algoritmo:

a) multiplicar o dígito mais significativo por B_o;

b) somar ao resultado o algarismo seguinte (à direita, é claro), obtendo um novo resultado;

c) multiplicar esse novo resultado por B_o;

d) repetir o processo até se atingir o último algarismo à direita (o menos significativo).

Exemplo A.12

Converter o número $1\ 0\ 1\ 1\ 1\ 1\ 0_2$ para a base 10.

$1 \times 2 + 0\ (d_5) = 2$

$2 \times 2 + 1\ (d_4) = 5$

$5 \times 2 + 1\ (d_3) = 11$

$11 \times 2 + 1\ (d_2) = 23$

$23 \times 2 + 1\ (d_1) = 47$

$47 \times 2 + 0\ (d_0) = 94$

O resultado é: $9\ 4_{10}$.

Exemplo A.13

Converter $3\ 1\ 2\ 5_8$ para decimal.

$3 \times 8 + 1 = 25$

$25 \times 8 + 2 = 202$

$202 \times 8 + 5 = 1621$

O resultado é: $1\ 6\ 2\ 1_{10}$.

Um outro método de conversão de bases consiste na realização das operações estabelecidas pelo polinômio A.3, somente que, nesse caso, a aritmética refere-se à base do resultado, B_r.

Para tanto, torna-se necessário conhecer a forma de representação de cada elemento a ser operado, bem como efetuar as multiplicações e somas na base do resultado.

Exemplo A.14

Converter $3\ 7_{10}$ para a base 2.

$N = d_{n-1} \times B_o^{n-1} + d_{n-2} \times B_o^{n-2} + \ldots + d_0 \times B_o^0$

ou

$N = \{[(d_{n-1} \times B_o + d_{n-2}) \times B_o + d_{n-3}] + \ldots + d_1\} \times B_o + d_0$

$N = 3 \times 10^1 + 7 \times 10^0$

Na aritmética binária teremos:

$N = 11_2 \times 1010_2^1 + 111_2 \times 1010_2^0 =$

$= 11110_2 + 111_2 = 100101_2$

O resultado é: $1\ 0\ 0\ 1\ 0\ 1_2$.

Exemplo A.15

Converter $1\ 8\ 9_{10}$ para a base 8.

$N = [(d_{n-1} \times B_o + d_{n-2}) \times B_o] + d_0$

onde:

$d_{n-1} = d_2 = 1_{10} = 1_8$

$d_{n-2} = d_1 = 8_{10} = 10_8$

$d_0 = d_0 = 9_{10} = 11_8$

$B_o = 10_{10} = 12_8$

$n = 3$ (quantidade de algarismos do número)

$N_8 = [(1 \times 12 + 100 \times 12] + 11 =$

$= [(12 + 10) \times 12] + 11 = [22 \times 12] + 11 =$

$= 264 + 11 = 275_8$

Para conferir:

$275_8 = 2 \times 8^2 + 7 \times 8^1 + 5 \times 8^0 = 128 + 56 + 5 = 189_{10}$

A implementação de um algoritmo para a conversão com essa aritmética (de B_o) requereria a prévia determinação de tabelas de soma e multiplicação na base desejada, de modo semelhante ao mostrado na Fig. A.1.

Para valores fracionários, o processo é semelhante, usando-se a expressão:

$N = B_o^{-1} \times \{d_{-1} + B_o^{-1} \times [d_{-2} + \ldots + B_o^{-1} \times (d_{-(m-1)}) + B_o^{-1} \times d_{-m}] \ldots\}$

obtida de:

$$N = d_{-1} \times B^{-1} + d_{-2} \times B^{-2} + d_{-3} \times B^{-3} + \ldots + d_{-m} \times B^{-m}$$

sendo **m** a quantidade de algarismos fracionários do número.

Até esse ponto foram apresentados algoritmos de conversão de números de uma base B_o para outra base B_d, considerando-se o caso mais geral, onde B_o e B_d podem assumir quaisquer valores inteiros e positivos (exceto os valores 0 e 1), inclusive 10.

No entanto, sabemos que:

a) todo computador digital é construído para funcionar internamente apenas com valores binários;

b) como estamos acostumados a trabalhar no sistema decimal, os valores numéricos introduzidos no sistema computacional estão, usualmente, representados em base 10;

c) há ocasiões em que se torna necessário usar a linguagem binária interna da máquina (seja para introduzir valores diretamente nessa linguagem, seja para verificar ou analisar o conteúdo de sua memória). Para evitar trabalhar diretamente com números binários (possuem muitos algarismos), costuma-se converter esses valores internos para outros em base maior (pois quanto maior o vàlor da base, menor a quantidade de algarismos necessários para representar um número).

A maioria dos fabricantes vem empregando bases de valor igual ao de uma potência de 2, mais especificamente as bases 8 (octal) e 16 (hexadecimal), em virtude da rapidez da conversão, comparativamente ao que seria necessário para converter para outras bases diferentes das potências de 2.

No Cap. 2 (subitem 2.3.1), foram descritos os aspectos essenciais dessas conversões, sendo portanto desnecessário repetir os algoritmos.

Conversão através de código binário - decimal

(Binary Coded Decimal — BCD)

Uma das formas mais comuns de representação de algarismos em um computador consiste no método chamado BCD — Binary Coded Decimal, assim denominado pela estreita relação que faz entre algarismos decimais e seus valores em base 2.

Por esse método, cada algarismo decimal é convertido para um valor binário, com quatro dígitos (ou bits), segundo a relação:

0 → 0000
1 → 0001
2 → 0010
3 → 0011
4 → 0100
5 → 0101
6 → 0110
7 → 0111
8 → 1000
9 → 1001

O método BCD é empregado nos processos de Entrada e Saída (E/S), para posterior conversão em valores binários diretos. Ou seja, o número é convertido, na entrada (dispositivo ou unidade controladora de entrada), de base 10 para o valor correspondente em BCD; em seguida, é convertido para o valor binário direto.

Exemplo A.16

Decimal	BCD	Binário direto
17	0 0 0 1 0 1 1 1	1 0 0 0 1
35,4	0 0 1 1 0 1 0 1, 0 1 0 0	1 0 0 0 1 1, 0 1 1

Nas operações de saída o processo se inverte, isto é, o valor interno (binário direto) é primeiramente convertido para a forma BCD e, para apresentação no dispositivo de saída, é convertido para sua forma decimal (entendida pelo operador).

Além disso, há computadores capazes de efetuar operações aritméticas com números representados na forma BCD, de modo que os cálculos são executados como se fossem operações em decimal (embora os algarismos estejam representados em binário). No item 6.4 essas operações aritméticas são apresentadas em detalhe.

A.7 OPERAÇÕES ARITMÉTICAS

Todo sistema de computação moderno é construído de modo a ser capaz de armazenar, interpretar e manipular informações codificadas na forma binária. Além disso, muitos deles possuem a capacidade de representar valores e efetuar operações aritméticas utilizando recursos de outras bases da potência de 2 (mais especialmente as bases octal — base 8 e hexadecimal — base 16). Esse é o caso, por exemplo, da representação e aritmética de números em ponto flutuante (ver Cap. 6 — Unidade Central de Processamento); alguns sistemas de computação IBM empregam a base 16 quando efetuam aritmética em ponto flutuante.

A seguir serão descritos procedimentos para execução das quatro operações aritméticas de números binários (base 2), inteiros e fracionários, sem sinal (operações aritméticas de números com sinal serão apresentadas no Cap. 7 — Representação de Dados).

A título de ilustração, descreveremos também algumas operações em bases não-binárias; adição e subtração nas bases octal e hexadecimal, com os mesmos tipos de números.

A.7.1 Procedimentos de Adição

Tendo em vista que toda representação de valores nos computadores digitais é realizada no sistema binário, é óbvio, então, que as operações aritméticas efetuadas pela máquina sejam também realizadas na mesma base de representação, a base 2.

As operações de adição nas bases 2, 8 e 16 são realizadas de modo idêntico ao que estamos acostumados a usar para a base 10, exceto no que se refere à quantidade de algarismos disponíveis (que, em cada base, é diferente). Esse fato acarreta diferença nos valores encontrados, mas não no modo como as operações são realizadas.

A soma de dois números com sinal depende, em computadores, do modo de representação interna desses números; no Cap. 6 — Unidade Central de Processamento, foram descritos processos para realização de operações aritméticas entre números positivos e negativos; nessa parte, trataremos apenas de operações com valores sem sinal (inteiros e fracionários).

A.7.1.1 Adição de Números Binários

A operação de soma de dois números em base 2 é efetuada de modo semelhante à soma decimal, levando-se em conta, apenas, que só há dois algarismos disponíveis (0 e 1). Assim:

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$, com "vai 1"

Exemplo A.17

1 11111 1	"vai 1"	11 11 1
101101,01	parcela 1	11001,1101
+ 100111,11	parcela 2	+ 11100,111
1010101,00		110110,1011

Do mesmo modo que operamos na base decimal, a soma é efetuada algarismo por algarismo, de maneira que, quando somamos 1 com 1, obtemos como algarismo resultante 0 e sobra o valor 1 para ser somado aos algarismos da parcela imediatamente seguinte à esquerda (valor de uma base - 2); esse é o valor que denominamos "vai 1". Se os dois algarismos a serem somados são de valor igual a 1, e ainda temos o "vai 1" anterior a ser acrescentado, o algarismo resultante é igual a 1, com outro "vai 1" para o algarismo da esquerda.

Resumindo:

1 + 1 + 1 = 1 com "vai 1";

1 + 0 + 1 = 0 com "vai 1".

A.7.1.2 Adição de Números Octais e Hexadecimais

Os procedimentos para adição nas bases 8 (octal) e 16 (hexadecimal) também não diferem da base 10, exceto quanto à quantidade de algarismos diferentes em cada base, conforme já mencionamos anteriormente.

No caso da base octal, temos 7 algarismos disponíveis e, portanto, a soma de 2 algarismos produzindo um valor superior a 7 implica a utilização do conceito de "vai 1" (nessa base, o "vai 1" consiste em um valor igual a 8 na ordem inferior).

Para a base 16, o "vai 1" somente ocorre quando a soma de 2 algarismos excede o valor da base, 16.

Exemplo A.18

Soma com aritmética em base 8:

	(1)	(2)	(3)
"vai 1"	11	1 1 1	111 1
parcela 1	3463	422,74	27,416
parcela 2	+ 1524	+ 513,74	+ 55,635
resultado	5207	1136,70	105,253

A execução detalhada do algoritmo de soma para o exemplo (1) é apresentada a seguir, de modo que se possa compreender melhor o processo:

1) 3 + 4 = 7, valor colocado na coluna, em resultado;

2) 6 + 2 = 8 (não há esse algarismo na base 8 — o maior algarismo é 7). Assim, temos: 8 = 8 + 0; o 0 é colocado na coluna como resultado e o 8 é passado para a esquerda com valor 1 (é o "vai 1"), pois 8 unidades de uma ordem representam apenas 1 unidade de ordem superior — mais à esquerda;

3) 4 + 5 + 1 ("vai 1") = 10 (10 = 8 + 2; logo, é 2 na coluna de resultado e "vai 1" à esquerda, representando o valor 8);

4) 3 + 1 + 1 ("vai 1") = 5, colocado na coluna resultado.

O processo é semelhante para os exemplos (2) e (3).

Exemplo A.19

Soma com aritmética em base 16:

"vai 1"	11 1
primeira parcela	3A54,3B
+	
segunda parcela	1BE8,7A
soma	563C,B5

a) 10(A) + 11(B) = 21, que excede 5 da base 16. Logo, coloca-se 5 na linha "soma" e "vai 1" para a esquerda;

b) 7 + 3 + 1 = 11 (algarismo B);

c) 8 + 4 = 12 (algarismo C);

d) 14(E) + 5 = 19, que excede de 3 a base 16. Logo, coloca-se 3 na linha "soma" e "vai 1" para a esquerda;

e) 10(A) + 11(B) + 1 = 22, que excede 6 da base 16 e "vai 1";

f) 1 + 3 + 1 = 5.

A.7.2 Procedimentos de Subtração

Os procedimentos para execução da operação de subtração em bases 2, 8, 16, ou qualquer outra não-decimal seguem as mesmas regras adotadas para a base 10, variando, conforme já mencionado diversas vezes, quanto à quantidade de algarismos existentes em cada base.

Essas regras são:

1) minuendo − subtraendo = diferença;

2) operação realizada algarismo por algarismo;

3) se o algarismo do minuendo for menor que o algarismo do subtraendo, adiciona-se ao minuendo um valor igual ao da base (2 ou 8 ou 16). Esse valor corresponde a uma unidade subtraída (empréstimo) do algarismo à esquerda do minuendo;

4) o resultado é colocado na coluna, na parcela diferença.

Inicialmente trataremos da subtração binária e, no item seguinte, apresentaremos alguns exemplos de subtração nas bases 8 e 16.

A.7.2.1 Subtração de Números Binários

A subtração em base 2, na forma convencional usada também no sistema decimal (minuendo − subtraendo = diferença), é relativamente mais complicada por dispormos apenas dos algarismos 0 e 1. Assim, 0 menos 1 necessita de um "empréstimo " de um valor igual à base (no caso é 2), obtido do primeiro algarismo diferente de zero, existente à esquerda. Se estivéssemos operando na base decimal, o "empréstimo" seria de valor igual a 10.

Exemplo A.20

022	empréstimo	112012
101101	minuendo	100010001
− 100111	subtraendo	− 010101100
000110		001100101

Exemplo A.21

$$\begin{array}{r} 1001,101 \\ -\ 110,110 \\ \hline 0010,111 \end{array} \qquad\qquad \begin{array}{r} 101110000,00110 \\ -\ 10101101,01111 \\ \hline 11000010,10111 \end{array}$$

A.7.2.2 Subtração de Números Octais e Hexadecimais

Os procedimentos para realização da operação de subtração com valores representados em base 8 (octal) ou 16 (hexadecimal) são os mesmos das bases 2 ou 10, porém com a já conhecida diferença em termos de algarismos disponíveis.

Exemplo A.22

Subtração com valores em base 8:

$$\begin{array}{r} 3526,53 \\ -\ 2764,36 \\ \hline 0742,15 \end{array} \qquad\qquad \begin{array}{r} 46234,710 \\ -\ 15573,523 \\ \hline 30441,165 \end{array}$$

Exemplo A.23

Subtração com valores em base 18:

$$\begin{array}{r} 49AB,8D5 \\ -\ FC8,AB8 \\ \hline 39E2,E1D \end{array} \qquad\qquad \begin{array}{r} 54CF,6BC \\ -\ 379E,6FB \\ \hline 1D30,FC1 \end{array}$$

A.7.3 Multiplicação de Números Binários

O processo de multiplicação binária é realizado na forma usualmente efetuada para a base 10, isto é, somas sucessivas, visto que os algarismos do multiplicador somente podem ser 0 ou 1.

Exemplo A.24

$$\begin{array}{rl} 1011 & \text{multiplicando} \\ \times\ 101 & \text{multiplicador} \\ \hline 1011 & \text{primeiro produto parcial} \\ +\ 0000 & \text{segundo produto parcial} \\ 1011 & \text{terceiro produto parcial} \\ \hline 110111 & \text{produto final} \end{array}$$

Em um sistema numérico posicional, o deslocamento (shift) de um número uma posição para a esquerda ou para a direita (shift left ou shift right) é equivalente, respectivamente, à multiplicação ou divisão desse número pela sua base.

Por exemplo, na base 10, tendo o número:

1035, onde * 10 = 10350 ' (deslocamento à esquerda)

←

/ 10 = 0103,5 (deslocamento à direita)

→

Na base 2 teríamos:

101110, onde * 2 = 1011100 (deslocamento à esquerda)

←

/ 2 = 010111 (deslocamento à direita)

→

No caso da aritmética binária, a multiplicação se resume apenas em *deslocamento e soma*, conforme mostrado no exemplo anterior.

Poder-se-ia, também, efetuar a multiplicação através da soma sucessiva do multiplicando com ele mesmo, tantas vezes quanto fosse o valor do multiplicando.

É claro que nesses exemplos não estamos levando em consideração diversos fatores, tais como: o sinal do número (que pode alterar os resultados), o espaço disponível para armazenar os números internamente na máquina, etc.

A.7.4 Divisão de Números Binários

Como nas demais operações aritméticas, a divisão binária é efetuada de modo semelhante à divisão decimal, considerando-se apenas que:

0 / 1 = 0 e 1 / 1 = 1

e que a divisão por zero acarreta erro.

Podemos efetuar uma divisão binária pelo método comum, isto é, dividendo / divisor = quociente e resto. Ou podemos realizá-la através de sucessivas subtrações, um processo mais simples de implementação em circuitos digitais.

Nesse caso, o desejado quociente será a quantidade de vezes que o divisor poderá ser subtraído do dividendo, até que se obtenha um quociente igual a zero.

O outro método consiste na execução do algoritmo a seguir apresentado, o qual é o detalhamento do processo usado para executarmos essa operação no lápis e papel, na base decimal.

a) a partir da esquerda, avançam-se tantos algarismos quantos sejam necessários para obter-se um valor igual ou maior que o divisor;
b) encontrado esse valor, registra-se 1 para o quociente;
c) subtrai-se do valor obtido no dividendo o valor do divisor (na divisão binária, como o quociente somente pode ser de valor igual a 1, a subtração é sempre com o próprio valor do divisor);
d) ao resultado acrescentam-se mais algarismos do dividendo (se ainda houver algum), até obter-se um valor igual ou maior que o divisor (como no item *a*). Se o(s) algarismo(s) for(em) zero, acrescentam-se zero(s) ao quociente;
e) repete-se o processo a partir do item *b*, até que se esgotem os algarismos do dividendo.

Exemplo A.25

$100_2 / 10_2 = 10_2$ ou $4_{10} / 2_{10} = 2_{10}$

$$\begin{array}{r|l} 10'0 & \underline{10} \\ \underline{-10} & 10 \\ 00 & \end{array}$$

item a: 10 (dividendo) = 10 (divisor)

item b: quociente = 1

item c: 10 − 10 = 0

item d: dividendo = 00, quociente = 0

ou $18_{10} / 3_{10} = 6_{10}$

Exemplo A.26

$10010_2 / 11_2 = 110_2$ ou $18_{10} / 3_{10} = 6_{10}$

$$\begin{array}{l|l} 100'1'0 & 11 \\ \cline{2-2} -11 & 110 \\ \cline{1-1} 011 & \\ \;\;00 & \end{array}$$

Apêndice B

Códigos de Representação de Caracteres

Decimal	Hexadecimal	Caracteres		Decimal	Hexadecimal	Caracteres	
		ASCII	**EBCDIC**			**ASCII**	**EBCDIC**
000	00	NUL	NUL	033	21	!	SOS
001	01	SOH	SOH	034	22	“	FS
002	02	STX	STX	035	23	#	
003	03	ETX	ETX	036	24	$	BYP
004	04	EOT	PF	037	25	%	LF
005	05	ENQ	HT	038	26	&	ETB
006	06	ACK	LC	039	27	‘	ESC
007	07	BEL	DEL	040	28	(	
008	08	BS		041	29	)	
009	09	HT	RLF	042	2A	*	SM
010	0A	LF	SMM	043	2B	+	CU2
011	0B	VT	VT	044	2C	‘	
012	0C	FF	FF	045	2D	-	ENQ
013	0D	CR	CR	046	2E	.	ACK
014	0E	SO	SO	047	2F	/	BEL
015	0F	SI	SI	048	30	0	
016	10	DLE	DLE	049	31	1	
017	11	DC1	DC1	050	32	2	SYN
018	12	DC2	DC2	051	33	3	
019	13	DC3	TM	052	34	4	PN
020	14	DC4	RES	053	35	5	RS
021	15	NAK	NL	054	36	6	UC
022	16	SYN	BS	055	37	7	EOT
023	17	ETB	IL	056	38	8	
024	18	CAN	CAN	057	39	9	
025	19	EM	EM	058	3A	:	
026	1A	SUB	CC	059	3B	;	CU3
027	1B	ESC	CU1	060	3C	<	DC4
028	1C	FS	IFS	061	3D	=	NAK
029	1D	GS	IGS	062	3E	>	
030	1E	RS	IRS	063	EF	?	SUB
031	1F	US	IUS	064	40	@	SP
032	20	SP	DS	065	41	A	

Decimal	Hexadecimal	Caracteres		Decimal	Hexadecimal	Caracteres	
		ASCII	EBCDIC			ASCII	EBCDIC
066	42	B		122	7A	z	:
067	43	C		123	7B	{	#
068	44	D		124	7C	|	@
069	45	E		125	7D	}	‘
070	46	F		126	7E	~	=
071	47	G		127	7F	DEL	“
072	48	H		128	80	NUL	
073	49	I		129	81		a
074	4A	J	[	130	82		b
075	4B	K	.	131	83		c
076	4C	L	<	132	84		d
077	4D	M	(	133	85		e
078	4E	N	+	134	86		f
079	4F	O	|	135	87	BEL	g
080	50	P	&	136	88	BS	h
081	51	Q		137	89	HT	i
082	52	R		138	8A	LF	
083	53	S		139	8B	VT	
084	54	T		140	8C	FF	
085	55	U		141	8D	CR	
086	56	V		142	8E	SO	
087	57	W		143	8F	SI	
088	58	X		144	90		
089	59	Y		145	91	DC1	j
090	5A	Z	]	146	92	DC2	k
091	5B	[	$	147	93	DC3	l
092	5C	\	*	148	94	DC4	m
093	5D	]	)	149	95		n
094	5E	^	;	150	96		o
095	5F	_		151	97		p
096	60	‘	–	152	98	CAN	q
097	61	a	/	153	99		r
098	62	b		154	9A		
099	63	c		155	9B	ESC	
100	64	d		156	9C		
101	65	e		157	9D		
102	66	f		158	9E		
103	67	g		159	9F		
104	68	h		160	A0	á	
105	69	i		161	A1	í	–
106	6A	j		162	A2	ó	s
107	6B	k	‘	163	A3	ú	t
108	6C	l	%	164	A4	ñ	u
109	6D	m	_	165	A5	Ñ	v
110	6E	n	>	166	A6	ª	w
111	6F	o	?	167	A7	º	x
112	70	p		168	A8		y
113	71	q		169	A9		z
114	72	r		170	AA		
115	73	s		171	AB		
116	74	t		172	AC		
117	75	u		173	AD		
118	76	v		174	AE		
119	77	w		175	AF		
120	78	x		176	B0		
121	79	y		177	B1		

Decimal	Hexadecimal	Caracteres		Decimal	Hexadecimal	Caracteres	
		ASCII	EBCDIC			ASCII	EBCDIC
178	B2			218	DA		
179	B3			219	DB		
180	B4			220	DC		
181	B5			221	DD		
182	B6			222	DE		
183	B7			223	DF		
184	B8			224	E0		
185	B9			225	E1		
186	BA			226	E2		S
187	BB			227	E3		T
188	BC			228	E4		U
189	BD			229	E5		V
190	BE			230	E6		W
191	BF			231	E7		X
192	B0			232	E8		Y
193	C1		A	233	E9		Z
194	C2		B	234	EA		
195	C3		C	235	EB		
196	C4		D	236	EC		
197	C5		E	237	ED		
198	C6		F	238	EE		
199	C7		G	239	EF		
200	C8		H	240	F0		0
201	C9		I	241	F1		1
202	CA			242	F2		2
203	CB			243	F3		3
204	CC			244	F4		4
205	CD			245	F5		5
206	CE			246	F6		6
207	CF			247	F7		7
208	D0		J	248	F8		8
209	D1		K	249	F9		9
210	D2		L	250	FA		
211	D3		M	251	FB		
212	D4		N	252	FC		
213	D5		O	253	FD		
214	D6		P	254	FE		
215	D7		Q	255	FF		
216	D8		R				
217	D9						

Apêndice C

Glossário

A

Acesso direto — Ato realizado pelo sistema para localizar uma posição de memória sem necessidade de acessos preliminares a outras posições. Este método é particularmente utilizado por discos magnéticos (HD — hard disks).

Acesso direto à memória (DMA) — Método de transferência de dados entre memória em disco e MP, pelo qual a transferência é realizada sem a interveniência da UCP, sendo realizada diretamente entre os dois dispositivos de memória por controle de um elemento de hardware denominado controlador de DMA.

Acesso seqüencial — Ato realizado pelo sistema para localizar uma posição de memória (qualquer tipo de memória), cujo endereço é identificado pela sua posição relativa em relação ao primeiro endereço de memória e que somente pode ser feito mediante acesso prévio às posições anteriores.

Acumulador (ACC) — Nome de um registrador especial da UCP, usado em certos processadores, para servir de elemento de armazenamento intermediário em instruções de um operando.

AGP — Accelerated Graphics Port. Trata-se de uma arquitetura de barramento de 32 bits, desenvolvida pela Intel e introduzida no mercado em 1997, sendo específica para interligação de placas de vídeo (interfaces) diretamente ao barramento do sistema.

Arquivo — Conjunto de dados (pode ser também um programa em código-fonte, código-objeto ou código executável) armazenado em memória secundária, identificado por um nome único para todo o conjunto.

ASCII — American Standard Code for Information Interchange. É um código de representação de caracteres (letras, algarismos, sinais de pontuação, operadores aritméticos, caracteres especiais de controle, etc.) com 7 bits para cada caractere. Os sistemas de microcomputadores empregam a versão de 8 bits desse código.

B

Backup — Termo inglês, tão popular em informática que o usamos como se pertencesse à nossa língua. Consiste na obtenção de uma cópia de um arquivo em um meio de armazenamento separado do original, com o propósito de segurança de dados, de forma que, se o arquivo original for apagado ou destruído acidentalmente, tem-se a cópia de backup para utilização.

Barramento — Um elemento crucial do sistema de computação, constituído de linhas de transmissão por onde os sinais elétricos (bits e sinais de controle) fluem entre o processador e demais componentes do sistema. Os barramentos (denominados em inglês *bus*) podem conduzir dados, endereços ou sinais de controle. Atualmente há diferentes tipos desses barramentos, cada um com sua especificação própria e aplicação definida. Entre esses, podemos citar ISA, EISA, PCI, AGP.

Barramento de controle — Elemento de interligação UCP/MP, por onde são conduzidos os sinais de controle e de sincronização. Internamente na UCP ele está conectado à Unidade de Controle, de onde são emitidos os citados sinais.

Barramento de dados — Elemento de interligação UCP/MP, por onde passam os bits que constituem a informação que está sendo transferida (podem ser dados ou instruções de máquina) entre os componentes.

Barramento de endereços — Elemento de interligação UCP/MP, por onde passam os bits que constituem o endereço de acesso a uma posição (célula) da MP.

Base — Elemento principal de um sistema de numeração posicional. Define a quantidade de algarismos existentes em um sistema, como 10 no sistema decimal (algarismos de 0 até 9) ou 2 no sistema binário (algarismos 0 e 1). O valor de um algarismo nos sistemas posicionais é variável; depende de sua posição relativa no número.

Base mais deslocamento — Modo de endereçamento empregado em alguns sistemas de computação. Neste modo, cada endereço de acesso é calculado pela UCP através da soma de dois valores: do conteúdo de um registrador (registrador-base) e do campo deslocamento existente na instrução.

BCD — Binary Coded Decimal. Código de representação de caracteres que usa seis bits para cada símbolo codificado (atualmente este código está em desuso). Também é um sistema de representação de números que redunda em uma aritmética especial em certos sistemas de computação.

BIOS — Basic Input-Output System (sistema básico de entrada e saída). Trata-se de um conjunto de programas elaborados em linguagem de baixo nível, armazenados em uma memória do tipo ROM (não-volátil) e que são usados pelos programas aplicativos e pelo sistema operacional para algumas funções de entrada e saída, como ler e interpretar um caractere do teclado, enviar um caractere para a impressão ou para o vídeo. Possui, também, os programas de inicialização do computador (POST).

Bit — Binary Digit. Em um sistema de representação binário (base 2) significa o algarismo 0 ou o algarismo 1. É a menor unidade de armazenamento e informação em um computador.

Bit de paridade — Bit (dígito binário) acrescentado a um grupo de bits (de uma célula ou de uma palavra de dados) de modo que o total de bits de valor 1 seja sempre par (paridade par) ou ímpar (paridade ímpar). Este procedimento permite verificar, em uma transferência do grupo de bits, se ocorreu algum erro.

Bloco (registro físico) — Unidade de transferência de dados entre MP e MS, usada em certos sistemas de grande porte. Pode conter um ou mais registros lógicos, acrescidos de informações de controle geradas pelo sistema de E/S para identificar o específico registro físico e permitir sua transferência.

Bps — Bits por segundo. Taxa de velocidade de transferência de dados em um circuito de comunicação, exprimindo a quantidade de bits que o equipamento transmissor e receptor pode operar. Um modem, que opera em 56 Kbps, transfere até cerca de 56.000 bits em cada segundo.

Byte — Conjunto formado por 8 bits consecutivos. Também chamado de octeto. Em geral, constitui-se de uma unidade de armazenamento na MP; grande quantidade de sistemas utiliza o byte como unidade de armazenamento e de transferência de dados pelos barramentos.

C

Cache — Tipo de memória de alta velocidade, inserida entre o processador e a memória principal, com a finalidade de aumentar o desempenho do sistema, visto que a velocidade processador/cache é muito maior do que processador/memória principal. Atualmente, os processadores possuem mais de um tipo de memória cache, denominadas cache nível 1 (L1) e cache nível 2 (L2), sendo a L1 inserida no interior da pastilha do processador, funcionando com a mesma velocidade, enquanto a L2 é normalmente localizada na placa-mãe.

Célula — Unidade de organização de armazenamento da MP. A memória principal é organizada como um conjunto de N células, cada uma podendo armazenar informação com um tamanho fixo e igual de M bits; uma célula tem, então, um tamanho de M bits. O grupo de bits que constitui uma célula de memória é transferido de uma vez em cada acesso (na realidade, são transferidas várias células em cada acesso).

Chipset — Conjunto de circuitos integrados inseridos em uma única pastilha, com a finalidade de executar uma variedade de funções durante a realização de um ciclo de instrução.

Ciclo de instrução — Conjunto de etapas previamente programadas na Unidade de Controle (por microprogramação ou diretamente por arranjo lógico no hardware) que permite o acesso à MP para a busca de uma instrução, sua interpretação (decodificação do código de operação) pela UCP e a conseqüente execução.

Ciclo de máquina — Também chamado de ciclo de memória. Conjunto de etapas previamente determinadas que permite a localização de uma célula de MP e o acesso a ela para armazenar ou recuperar um valor binário.

Cilindro — Unidade de armazenamento e transferência de dados armazenados em um disco magnético. É constituído do espaço de armazenamento compreendido por todas as trilhas de igual endereço em discos com múltiplas superfícies magnéticas.

Circuito integrado — Reduzido pedaço de material onde são armazenados (através de um processo complexo de fabricação) muitos componentes eletrônicos (resistores, transistores, capacitores, etc.) e suas interligações.

CMOS — Complementary Metal Oxide Semiconductor. Tecnologia de fabricação de pastilhas, segundo um processo pelo qual dispositivos N-channel e P-channel são dispostos de forma complementar entre si, de modo a obter um espaço menor e baixo consumo de energia.

Código de caracteres — Código numérico binário organizado para representar cada caractere a ser manipulado em um sistema de computação. O código ASCII, por exemplo, representa o caractere *t* como: 01110100, enquanto o código EBCDIC representa o mesmo caractere como: 10100011.

Código de correção de erros — Código semelhante ao anterior, exceto pelas regras de organização do conjunto de bits a ser transmitido, que são elaboradas de modo a permitir não só a verificação da ocorrência de erros, como também quais bits estão errados, para que a correção possa ser imediata pelo próprio receptor.

Código de operação — Op. Cod. ou C. Op. Grupo de bits em uma instrução de máquina que indica a operação a ser realizada pelo processador.

Complemento — Complemento de um número é um outro número, obtido de modo a completar o valor do número de origem. Na matemática há dois tipos de complemento: complemento à base e complemento à base menos um. No caso da base 2, denomina-se complemento a 2 e complemento a 1.

Contador de instrução — CI. Em inglês, denomina-se PC — Program Counter. É um registrador da UCP cuja função é armazenar o endereço da próxima instrução a ser buscada da MP e executada.

D

Decodificador de instrução — Elemento controlado pela UC, cuja função é receber um código de operação de uma instrução e produzir os sinais de controle adequados para sua execução.

Desvio — Alteração da seqüência de execução de uma instrução por meio da substituição do endereço atual armazenado no CI pelo endereço de desvio. Esta ação é realizada por uma instrução específica para esta finalidade. Pode haver dois tipos de instrução de desvio: desvio incondicional, onde o desvio é sempre realizado, e desvio condicional, no qual a ação de desvio só é executada pelo processador se uma determinada condição for verdadeira, como, por exemplo, se o ACC = 0.

DIMM — Dual In-line Memory Module. É uma forma de encapsulamento de pastilha, utilizado pelas memórias mais modernas, em substituição ao tipo SIMM.

DIP — Dual In-line Package. Outro modo de encapsular uma pastilha, no qual os pinos são dispostos nos lados maiores da pastilha.

Disco magnético — Elemento circular, plano e estreito, contendo uma superfície magnetizável, na qual podem ser armazenados dados sob a forma de campos magnéticos.

Disquete — Também conhecido como floppy-disk. Elemento de construção e forma semelhante aos discos magnéticos, porém com características diferentes de formato físico, capacidade de armazenamento e taxa de transferência de bits.

DMA — Direct Memory Access. Trata-se de um processo de transferência de dados entre MP e discos magnéticos (ou outro dispositivo), no qual não há interveniência do processador, liberando-o para execução de outras tarefas, enquanto os dados são transferidos.

DRAM — Dynamic Random Access Memory. Trata-se do tipo de memória utilizado nas memórias principais do tipo leitura/escrita (read/write) dos atuais sistemas de computação. Uma DRAM armazena cada bit em um invólucro constituído por um capacitor e um transistor. Como o capacitor não mantém sua carga indefinidamente, há necessidade de recarregamentos periódicos (*refreshing*).

E

EBCDIC — Extended Binary Coded Decimal Interchange Code. É um código de representação de caracteres (qualquer tipo de caractere, como: letras, algarismos, sinais de pontuação, operadores aritméticos, caracteres especiais de controle, etc.) com largura de 8 bits para cada caractere. Desenvolvido e utilizado exclusivamente pela IBM em seus computadores de grande porte.

EEPROM — Electrically Erasable Programmable Read Only Memory. Trata-se de uma memória especial do tipo ROM (memória somente para leitura) que pode ser apagada e escrita eletricamente. Esta memória não é volátil e costuma ser empregada para armazenar os programas que constituem a BIOS de um sistema.

Endereçamento direto — Modo de endereçamento pelo qual o valor armazenado no campo operando de uma instrução representa o endereço da MP onde está localizado o dado.

Endereçamento, espaço de — Conjunto de células contíguas de memória utilizado em uma referência qualquer.

Endereçamento imediato — Modo de endereçamento pelo qual o valor armazenado no campo operando de uma instrução representa o próprio dado.

Endereçamento indexado — Modo de endereçamento pelo qual o endereço de acesso à MP é obtido pela soma do valor armazenado em um registrador da UCP (registrador indexador) e o valor contido em campo específico da instrução.

Endereçamento indireto — Modo de endereçamento pelo qual o valor armazenado no campo operando de uma instrução é o endereço de uma posição de MP, cujo conteúdo é o endereço do dado.

Endereçamento por registrador — Modo de endereçamento pelo qual o operando é obtido por meio do valor armazenado em um registrador da UCP. Se for direto, então o valor armazenado no registrador é o próprio valor do dado; se for indireto, o valor armazenado no registrador é um endereço da MP, onde está armazenado o dado.

Entrada/Saída — **E/S**. Em inglês é denominado Input/Output (I/O). Operação de transferência de bits entre algum dispositivo periférico, como o teclado, o vídeo, a impressora e o subsistema de processamento, UCP/MP.
EPROM — Erasable Programmable Read Only Memory. É uma pastilha de memória do tipo não-volátil e que funciona também somente para leitura (ROM), porém pode ser apagada e escrita, como as EEPROM. No entanto, o processo de apagamento dessas memórias é diferente, utilizando-se um feixe de luz ultravioleta e não um pulso elétrico.
E/S por programa — Método de realização de operações de E/S no qual a UCP comanda toda a operação por meio da realização de um programa específico para esta finalidade, não podendo realizar outra tarefa enquanto a operação de E/S não estiver concluída.

F

Firmware — Conjunto de microinstruções armazenadas em memória ROM (memória de controle) e que serve para interpretar cada instrução de máquina a ser executada pelo processador.
Fita magnética — Uma tira de material plástico coberta com material magnetizável e que permite o armazenamento de dados sob a forma de campos magnéticos. A tira é enrolada em carretéis.
Flash memory (ver Memória flash).
Flip-flop — É um circuito eletrônico constituído internamente de diversos componentes de lógica que permite o armazenamento de uma entre duas possíveis informações (0 ou 1) em uma máquina digital. O flip-flop também pode ser conhecido como um elemento biestável (pode assumir uma de duas posições) ou chave (pode estar ligado, bit 1, ou desligado, bit 0).
Form Factor — Significa o tamanho físico e o aspecto de um dispositivo. É usado com freqüência para representar o tamanho de circuitos impressos (placas). Por exemplo, com relação a discos, o form factor é o diâmetro de cada prato do disco (discos multipratos), como 3,5" ou 5,25".
FPU — Floating Point Unit (unidade de ponto flutuante). Trata-se do nome usualmente utilizado para uma unidade aritmética que realiza operações com dados somente representados em ponto flutuante. Atualmente, os processadores têm sido projetados com a UAL — unidade aritmética e lógica separada em unidades independentes para realizar operações com inteiros e com números em ponto flutuante.

G

Giga — Prefixo que representa um valor igual a $2^{30} = 1024M = 1024 \times 1024K = 1.073.741.824$

H

Hard disk (ver Disco magnético).
Hardware — Conjunto de equipamentos que constituem o computador. Trata-se da parte física do sistema, todos os seus elementos visíveis, partes mecânicas, elétricas, magnéticas, etc. Contrasta com a outra parte do sistema de computação, o software (ver definição neste Glossário).
Hz — Hertz. Unidade de medida de freqüência de sinais do tipo transmissão por variação eletromagnética (caso dos sinais elétricos que caminham no interior dos computadores, do som e da luz). Um Hz é equivalente a 1 pulso ou ciclo de variação do sinal por segundo.

I

IC — Integrated Circuit (circuito integrado). Trata-se de um conjunto de componentes eletrônicos e suas conexões que é produzido e encapsulado em um pequeno pedaço de material, como silício. Também conhecido como pastilha (ou chip, em inglês).
Instrução de máquina — Conjunto de bits que especificam uma operação a ser realizada e o valor ou a localização de um ou mais dados (operandos) que serão manipulados pela referida operação.
Interface de E/S — Conjunto de circuitos (hardware) e programas (software) que interligam um ou mais dispositivos de E/S e o subsistema UCP/MP para controlar e efetivar a transferência de bits entre esses elementos.
Interrupção — Trata-se de um evento ou sinal que acarreta a suspensão de um processo em execução, transferindo a atenção da UCP para um outro programa (em geral, uma rotina que identifica e interpreta o motivo da interrupção), e depois promove condições para o retorno da execução do processo suspenso (interrompido).
IRQ — Interrupt Request (solicitação de interrupção). É um sinal emitido por um dispositivo de hardware qualquer, como um teclado ou controlador de disco, indicando a necessidade de interrupção da operação atual da UCP (ver Interrupção). Os sinais IRQ de cada dispositivo caminham por linhas separadas, que o conectam a um componente controlador das interrupções e que, sendo programável, realiza sua identificação e providências a respeito.

K

Kilo — Prefixo que representa um valor igual a 2^{10} = 1024. Exemplo: 5K = 5 × 1024 = 5120.

L

Latência (ou latência rotacional). É o tempo médio para que um setor de uma trilha de um disco (os bits desse setor) passem sob a cabeça de leitura/escrita do disco, durante sua rotação, após ter sido completada a fase de busca (*seek*), onde o braço mecânico com a cabeça é posicionado sobre a trilha desejada.

Linguagem Assembly — Linguagem simbólica semelhante à linguagem de máquina (em geral há uma correspondência de um para um entre instruções Assembly e instruções de máquina), criada para facilitar a elaboração de programas em nível de máquina.

Linguagem de aplicação — Também chamada de linguagem de alto nível. Constituída de comandos e estruturas adequadas ao entendimento do programador na elaboração dos programas e sem direta relação com qualquer processador.

Linguagem de máquina — Linguagem que pode ser diretamente interpretada pelos circuitos internos da máquina (pelo hardware); trata-se da linguagem binária.

M

Mega (M) — Prefixo que representa um valor igual a 2^{20} = 1024K = 1024 × 1024 = 1.048.576. Exemplo: 4M = 4 × 1.048.576 = 4.194.304.

Memória cache — Tipo de memória de alta velocidade (compatível com a velocidade da UCP) e de capacidade menor que a da MP, localizada logicamente entre a UCP e a MP para armazenar dados ou instruções que deverão ser imediatamente utilizadas pela UCP. Usada com a finalidade de acelerar o processamento do sistema UCP/MP.

Memória EPROM (ver EPROM).

Memória flash — Apresenta as mesmas características de uma EEPROM (ROM não-volátil e apagável por um sinal elétrico), embora nas flash o apagamento não possa ser efetuado a nível de byte como nas EEPROM. O termo flash foi imaginado devido à elevada velocidade de apagamento dessas memórias em comparação com as antigas EPROM e EEPROM.

Memória principal — Tipo de memória, em geral de semicondutor, utilizada para armazenamento de instruções e dados de programas que serão ou estão sendo executados pela UCP. É considerada uma memória de trabalho da UCP, sendo organizada em células com tamanho fixo e igual, cada uma identificada por um número denominado endereço.

Memória PROM — (Programable Read Only Memory (memória ROM programável). Tipo de memória de semicondutor cujo conteúdo é armazenado uma vez após o processo de fabricação (a memória é construída "virgem", para posterior gravação dos bits), não podendo depois ser reutilizada.

Memória RAM — Random Access Memory (memória de acesso aleatório). Tipo de memória na qual qualquer posição é localizada por meio de um número que indica seu endereço (sua localização) e cujo tempo para esse acesso é o mesmo, independentemente de qual posição foi acessada anteriormente. Contrasta com um outro tipo de memória na qual o acesso é seqüencial. Com esse nome também é conhecida a parte da memória principal de um computador, que fica disponível para o usuário ler ou gravar seus programas e dados.

Memória ROM — Read Only Memory (memória somente para leitura). Tipo de memória de semicondutor, cujo conteúdo é armazenado durante o processo de fabricação da memória e que não pode ser depois alterado. A memória ROM não é reutilizável.

Memória secundária — Tipo de memória de grande capacidade, com a característica de ser permanente no armazenamento (não é volátil), usada para guardar informações (dados e programas) que não serão imediatamente usadas pela UCP. Discos, fitas, disquetes, CD-ROMs, são exemplos de memória secundária.

MHz — Megahertz. Unidade de medida de freqüência de sinais do tipo transmissão por variação eletromagnética (caso dos sinais elétricos que caminham no interior dos computadores, do som e da luz). Um MHz é equivalente a 1 milhão de pulsos ou de ciclos de variação do sinal por segundo.

Microcomputador — Conjunto formado por uma UCP, memória principal e dispositivos de entrada e saída, de pequeno porte, utilizados em geral por uma pessoa isoladamente.

Microinstruções — É o mais baixo nível de instruções que controla diretamente o funcionamento de um processador (quando ele é fabricado utilizando este método de controle, denominado microprogramação, diferente do método de controle denominado "por hardware", geralmente empregado em processadores com arquitetura tipo RISC).

Mícron — Unidade de medida de distância, muito empregada em processadores devido a seu diminuto valor. Um mícron é equivalente a um milionésimo do metro.

Microprocessador — UCP de pequenas dimensões, tendo todos os seus circuitos e componentes armazenados e interligados em um único invólucro, denominado pastilha (ou chip).
MIPS — Milhões de instruções por segundo. Trata-se de uma unidade de medida de desempenho, muito empregada em testes com processadores.
Multiprogramação — Modo de operação de um sistema de computação pelo qual diversos programas, armazenados na memória principal, compartilham o uso da UCP. Em face da grande velocidade de processamento da UCP, ela executa uma pequena parte de cada programa em um pequeno intervalo de tempo, desvia para executar parte de outro programa, até atingir o processamento do último, retornando para executar outra parte do primeiro programa e assim sucessivamente.

O

Operando — Dado que será manipulado por uma instrução. Sua localização (ou ele próprio) faz parte da instrução (campo do operando).

P

Palavra — Conjunto de bits cujo tamanho é decisão do fabricante do processador e que em geral está relacionado à capacidade de processamento da UCP. A Unidade Aritmética e Lógica (UAL) opera com valores do tamanho da palavra, bem como este é o tamanho dos registradores da UCP.
Pipelining — Processamento pipelining. Trata-se de uma técnica de organizar o processador e seu modo de funcionamento para execução do ciclo de instrução, que consiste em dividir o processador em módulos (ou estágios) que funcionam independentes e simultaneamente, de modo que é possível executar múltiplas instruções de cada vez, sendo que, em um dado instante de tempo, cada uma estará em uma fase (um estágio) diferente do ciclo.
Ponto fixo — Representação de um número em sistema posicional no qual o ponto fracionário (ou vírgula) é implicitamente assumido em uma posição fixa. Em computação, essa forma de representação permite três modalidades diferentes para se representar números negativos: sinal e magnitude (S/M), complemento a um (C1) e complemento a dois (C2).
Ponto flutuante — Tipo de representação de dados em um sistema de computação (correspondente à notação científica) no qual um número binário é representado por um grupo de três informações distintas: o sinal do número, uma parte fracionária composta dos dígitos significativos do número e o expoente, pelo qual uma base é sucessivamente multiplicada. O valor do número é igual ao produto da parte fracionária pela base elevada ao expoente indicado.
Porta lógica (gate) — Um circuito eletrônico que produz como saída um valor de sinal correspondente ao resultado de uma operação booleana sobre os sinais de entrada.

R

Registrador — Tipo de memória especial de mais alta velocidade em um computador. Em geral, os registradores encontram-se localizados na UCP, tais como: o RI, o CI, o REM, o ACC e outros de emprego mais geral.
Registrador de dados de memória (RDM) — Registrador que armazena um valor que acabou de ser copiado da MP, para ser transferido para a UCP ou um valor enviado pela UCP para ser escrito na MP.
Registrador de endereços de memória (REM) — Registrador que armazena o endereço de uma posição de MP que a UCP deseja acessar.
Registro físico — (ver Bloco).
Registro lógico — Conjunto de itens de dados que formam uma unidade de informação única (por exemplo, a matrícula, o nome, o departamento, o salário, etc. de uma pessoa, em um sistema de cadastro).
Relógio (clock) — Elemento responsável pela manutenção do sincronismo entre os diversos eventos constantes de um ciclo de instrução, bem como da velocidade com que esses eventos são realizados.

S

Setor — Parte do espaço de armazenamento de uma trilha em discos magnéticos ou disquetes. Em geral, discos ou disquetes têm cada trilha dividida em uma quantidade de setores, cada um permitindo o armazenamento de uma quantidade fixa e igual de bytes. É a menor unidade de transferência em discos e disquetes.
Sinal e magnitude — Modalidade para representação de dados em ponto fixo, na qual cada valor é representado como uma seqüência binária constituída de 1 bit para indicar o sinal do número e os restantes bits para indicar o valor do número (sua magnitude).
Software — Refere-se a todos os elementos de programação de um sistema de computação, isto é, todos os programas, sejam de aplicação ou básicos do sistema, contrastando com a parte física e visível do sistema, o hardware.

T

Tabela verdade — Tabela que descreve uma função lógica relacionando todas as possíveis combinações de valores de entrada e apresentando, para cada combinação, o valor em que a saída é verdadeira (ou falsa).
Tempo de acesso — Tempo gasto entre o início dos eventos que conduzem à leitura ou gravação (escrita) de uma informação entre UCP e MP e o instante em que a MP está pronta para receber nova solicitação (pode ser igual somente ao tempo gasto para interpretar o endereço e efetivar a operação, chamado de ciclo de máquina ou de memória, se esta for do tipo estática; ou pode ser igual ao tempo do ciclo de memória mais o tempo de refreshing da memória, se esta for do tipo dinâmica).
Tempo de busca (seek) — Tempo gasto pelo mecanismo de leitura e gravação de um sistema de disco magnético ou disquete para se posicionar sobre a trilha que será acessada (a partir da trilha em que estava posicionado quando foi solicitado este novo acesso).
Tempo de latência (ver Latência rotacional).
Trilha — É a parte de um disco ou fita magnética em que estão gravadas as informações. Constitui uma unidade de acesso e endereçamento nos discos magnéticos.

U

Unidade aritmética e lógica (UAL) — É a parte da UCP responsável pela efetiva execução das operações matemáticas sobre os dados, incluindo-se as aritméticas e as operações lógicas. Realizam-se, também, ações de comparação e testes de resultados.
Unidade central de processamento (UCP) — É o componente de hardware de um computador, constituído da Unidade Aritmética e Lógica (UAL), da Unidade de Controle (UC), dos registradores, dos barramentos internos para interligação entre os diversos componentes.
Unidade de controle — É a parte da UCP responsável pelas tarefas de interpretação das instruções, sincronização e controle da execução de todos os eventos do sistema.

V

Vetor — Estrutura de dados constituída de um conjunto de elementos individual e seqüencialmente armazenados, que podem ser acessados e manipulados individualmente através de um endereço referenciado por um índice.

Respostas dos Exercícios

CAPÍTULO 1

1) Embora os termos possam ser utilizados como sinônimos, *dado* é a matéria-prima para um processamento qualquer. Ou seja, um programa de computador processa os dados que lhe são fornecidos. A saída desse processamento é a informação. Nada impede que a informação produzida por um processamento seja utilizada como dados para outro processamento posterior.

2) a) Digitação do programa e preparação dos dados.
   b) Execução (processamento) → cálculos, testes etc.
   c) Geração de resultados → impressora ou monitor de vídeo.

3) Conjunto de partes coordenadas que concorrem para um determinado objetivo. O corpo humano parece ser um dos mais complexos e perfeitos sistemas. Uma grande empresa ou organização também pode ser encarada como um grande sistema, com entradas, processamento e saída.

4) Nível operacional: sistema para cadastramento de dados pessoais, sendo operado por digitadores profissionais.
   Nível gerencial: sistema que gera relatórios de vendas por setores na empresa.
   Nível estratégico: sistema para geração de balanços globais da empresa, para decisões em nível de diretoria.

5) É a formalização de um algoritmo em linguagem capaz de ser transformada em instruções precisas que serão, por sua vez, executadas por um computador, gerando os resultados apropriados. Por algoritmo entenda-se o conjunto de etapas necessárias à resolução de um problema qualquer.

6) Hardware: conjunto de componentes eletrônicos e eletromecânicos existentes num computador.
   Software: conjunto de programas e sua documentação (manuais etc.), de propósitos diversos, inseridos em um computador para nele serem processados.

7) Um conjunto de comandos, regras e facilidades que devem ser seguidos para se obter a mencionada formalização de um algoritmo. Serve para se expressar o problema a ser resolvido por uma máquina, mas perfeitamente definido e "entendível" do ponto de vista humano. Três exemplos: Pascal, C e Fortran.

8) Charles Babbage.

9) A utilização de uma máquina tabuladora mecânica, invenção de Herman Hollerith, que contava, classificava e ordenava dados previamente inseridos em cartões perfurados.

10) A quebra de códigos militares alemães secretos, por ocasião da Segunda Grande Guerra Mundial.

11) Foi o chamado 8008 e destinava-se a equipar calculadoras portáteis. Seu fabricante é a Intel Corporation que, até hoje, é um dos principais fabricantes de microprocessadores para computadores pessoais.

12) a) Instruções (o programa) e dados residem na mesma memória.
    b) Toda a informação (instruções e dados) residente na memória é acessada através do endereço das células que ocupa.
    c) As instruções estarão dispostas na memória de forma a serem executadas seqüencialmente.

CAPÍTULO 2

1) Componente de um Sistema de Computação que armazena não só o(s) programa(s) a ser(em) executado(s) pela UCP, como também as informações introduzidas por algum componente de ENTRADA (teclado, por exemplo) ou calculadas no decorrer da execução.

   Um exemplo é o cérebro humano, que executa, simultaneamente, as funções da UCP e da Memória. Ao se resolver a equação

   $X = (8 * 7) + (64 / 8)$

   "de cabeça", uma pessoa precisa:

   - Ler a equação e memorizá-la.
   - Executar as operações (como uma UCP) e armazenar os resultados intermediários (por exemplo, $8 * 7 = 56$).

   A equação (o programa) e os resultados intermediários são "armazenados" na nossa memória, da mesma forma que num SC.

   Outro exemplo está em equipamentos domésticos que "armazenam" programas a serem executados: videocassete e microondas, ou mesmo as agendas eletrônicas e telefones celulares, que armazenam informações e as apresentam quando necessárias.

2) Realizar as operações matemáticas com os dados e controlar quando e o que deve ser realizado pelos demais componentes.

3) Armazenar programas e dados para execução imediata (memória principal) ou para execução e uso posterior (memória secundária).

4) Para permitir que os Sistemas de Computação se comuniquem com o mundo exterior, convertendo, também, a linguagem do SC para a linguagem do meio exterior (caracteres e números).

5) Arquivos (cadastro) de pessoal, de clientes, de contas a pagar e faturas a receber.

6) Bit: menor unidade de informação armazenável; dígito binário.
   Byte: um conjunto de 8 bits.
   Palavra: unidade de transferência de informação entre a UCP e a Memória (por exemplo, palavra de 32 bits significa que 32 bits são transferidos, de cada vez, entre a UCP e a Memória).

7) Se $1K = 1024$, $1M = 1024*1024$, $1G = 1024*1024*1024$ e $1 \text{ byte} = 8 \text{ bits}$ então:

   a) $65.536 = xK$
      $x = 65.536/1024 = 64$

   b) $12.288K = xM$
      $x = 12.288 * 1024 / 1024 * 1024 = 12$

   c) $19.922.944 = xM$
      $x = 19.922.944 / 1024 * 1024 = 19$

   d) 8 Gbytes = x bytes
      $x = 8 * 1024 * 1024 * 1024 = 2^3 * 2^{30} = 2^{33}$

   e) 64 Kbytes = x bits
      $x = 64K * 8 \text{ bits} / \text{bits} = 512$

   f) 262.144 bits = xK bits
      $x = 262.144 \text{ bits}/1024 \text{ bits} = 256$

   g) 16.777.216 palavras = x palavras (usando a menor unidade possível)
      $x = 16.777.216 / 1024 / 1024 = 16M$

   h) 128G bits = x bits
      $x = 128 * 2^{30} = 2^7 * 2^{30} = 2^{37}$ bits

   i) 512K células = x células
      $x = 512 * 1024 = 524.288 = 2^{19}$

   j) 256 Kbytes = x bits
      $x = 256 * 1024 * 8 = 2^8 * 2^{10} * 2^3 = 2^{21} = 2M$

8) Vazão define a quantidade de transações que pode ser executada por um sistema na unidade de tempo (ex.: quantidade de atualizações em um sistema de controle de estoque). Tempo de resposta é uma medida ligada ao desempenho do sistema como um todo, e não dos componentes de *per se* (ex.: tempo entre a solicitação de saldo e a sua apresentação na tela).

9) Um supercomputador é projetado para executar grandes quantidades de cálculos matemáticos o mais rapidamente possível. Normalmente tem aplicação científica (simulações) ou estratégica (Pesquisa e Desenvolvimento em laboratórios governamentais).

10) Uso pessoal e UCP num único chip.

11) Linguagem de alto nível é bem próxima da linguagem do ser humano (tipicamente a língua inglesa) e por ele é compreendida. Linguagem de máquina é a linguagem que os computadores entendem, mas extremamente difícil de ser compreendida pelo ser humano. Pessoas programam em linguagens de alto nível e, depois, convertem seus programas para a linguagem de máquina, por um processo conhecido como *compilação* (ver Cap. 9).

12) Dados e instruções residem na mesma memória; os dados e instruções na memória são acessados pelo seu endereço; a execução das instruções ocorre de forma seqüencial.

13) Intel 80486, Intel Pentium III, AMD K6, Digital/Compac ALPHA e IBM RISC 6000.

CAPÍTULO 3

1) a) 101001101
b) 100011100
c) 111011001
d) 1000101
e) 10000111
f) 11010111
g) 1001000101
h) 11000101

2) a) 1770
b) 1645
c) 1039
d) 14
e) 3689
f) 4035
g) 2840
h) 2054

3) a) 261
b) 376
c) 160
d) 1317
e) 527
f) 33
g) 1465
h) 305

4) a) 110100001
b) 1110001
c) 1100110011
d) 1001101
e) 11111011
f) 1100000001
g) 10110100
h) 11011

5) a) 99
b) 1405
c) 1561
d) 45
e) 515
f) 30966
g) 180
h) 110

6) a) 1625
b) 1413
c) 1142
d) 1121
e) 505
f) 330
g) 635
h) 1011

7) a) 261
b) 319
c) 159
d) 38
e) 453
f) 123
g) 129
h) 298

8) a) 1BF
b) 220
c) DF
d) 47
e) 26E
f) 61
g) 79
h) 129

9) a) 930
b) 827
c) 1569
d) 153
e) 7892
f) 2031
g) 556
h) 4362

10) a) 1118
b) 993
c) 395
d) 358
e) 80
f) 207
g) 832
h) 953

11) a) 87D
b) 52D
c) 2E7
d) 166
e) 2A9
f) 3A9
g) 340
h) 687

12) a) 8615
b) 9993
c) 637
d) 3679
e) 9041
f) 6574
g) 44271
h) 8253

13) a) 3F11
b) 84539
c) 363
d) 537365
e) 101011011001
f) 433
g) 430
h) 100001011

14) a) 40707_8
b) 66535_8
c) 342418_{16}
d) 131_7
e) 10010110011_2
f) 333231_4
g) 6631_8
h) 1001111101

15) a) $3CC74_{16}$
b) 366_8
c) 44_6
d) 40422_5
e) 1001000011_2
f) 1011111011_2
g) $4DB1_{16}$
h) 00101_2

16) a) 010011001111_8
b) 00011010010001011011_2
c) 0011011001010001_2
d) 31333_8

17) a) 5710_8
b) $AAF9A4_{16}$
c) 11010110_2
d) 10110100100_8
e) 1031200_4
f) $E2B65_7$
g) $121C43_{16}$
h) 412_8

a) $110000001101_2 - 10110011101_2 =$
b) $35A3_{16} - 2FEC_{16} =$
c) $37425_8 - 14766_8 =$
d) $1001001_2 - 111100_2 =$

18) a) 11001110000_2
b) $5B7_{16}$
c) 22437_2
d) 0001101_2

19) B^n

20) 110100, 110101, 110110, 110111 e 111000.

21) 110000, 110011, 110110, 111001, 111100 e 111111.

22) 1366, 1367, 1370, 1371, 1372, 1373, 1374 e 1375.

23) 3746, 3750, 3752, 3754, 3756, 3760, 3762 e 3764.

24) 2BEFA, 2BEFB, 2BEFC, 2BEFD, 2BEFE, 2BEFF, 2BF00, 2BF01, 2BF02, 2BF03, 2BF04 e 2BF05.

25) 3A5B, 3A5F, 3A63, 3A67, 3A6B, 3A6F, 3A73, 3A77, 3A7B e 3A7F.

26) 1111111111 = 1023.

27) A = 3, B = 1, C = 0, D = 2.

28) A = 3, B = 5, C = 1, D = 0, E = 4, F = 2.

29) a) 164436 +177376 = 364034_8
b) EFC + DE2 = CDE_{16}
c) 180 + 512 = 692_{16}
d) 148 + 041 = 189_{16}
e) 9D + FE = $19B_{16}$
f) 37124 + 1257 = 40403_8
g) 2665 + 1376 = 4063_8
h) 2536 + 33367 = 36125_8

30) 1100100_2, 10201_3, 1210_4, 400_5, 244_6, 202_7, 144_8 e 121_9.

31) $3^4 = 81$.

32) $2^6 = 64$ números.

33) $8^3 = 512$.

34) $2^8 = 256$.

35) Maior número binário de 7 algarismos: $1111111_2 = 127$.

36) a) 7 b) 14
c) 8 d) 1001020_4 → **4**

37) 5ECFD, 5ECFE, 5ECFF, 5ED00, 5ED01 e 5ED02.

38) a) A400 b) A83C − A3FF = 043D

39) a) 17_9 b) 71_9 c) 707_9
d) 258_9 e) 871_9 f) 87_9

40) a) 136_6 b) 232_7 c) 138_9
d) 210122211_4 e) 3014_7 f) 233_5

41)

Decimal	Binário	Octal	Hexadecimal
37	100101	45	25
205	11001101	315	CD
238	011101110	356	EE
6732	0001101001001100	15114	1A4C
141	10001101	215	8D
117	1110101	165	75
10843	0010101001011011	25133	2A5B
303	100101111	457	12F

42) a) 100011_2 b) 100100010_2 c) 101001110110_2 d) 10010011100011 e) 1010 e resto = 1
f) 11011 e resto = 1 g) 100011111101001

43) 684_{10}

44) A relação matemática é de multiplicação (por 2, que é a base).

45) 11001 e 1111

CAPÍTULO 4

1) a) A.B.C+ not (A.B.C)

A	B	C	A . B	A . B . C	not (A.B.C)	A.B.C + not (A.B.C)
0	0	0	0	0	1	1
0	0	1	0	0	1	1
0	1	0	0	0	1	1
0	1	1	0	0	1	1
1	0	0	0	0	1	1
1	0	1	0	0	1	1
1	1	0	1	0	1	1
1	1	1	1	1	0	1

b) A.(notC + B + notD)

A	B	C	D	not C	notC+ B	not D	NotC + B+ notD	A.(notC + B+ notD)
0	0	0	0	1	1	1	1	0
0	0	0	1	1	1	0	1	0
0	0	1	0	0	0	1	1	0
0	0	1	1	0	0	0	0	0
0	1	0	0	1	1	1	1	0
0	1	0	1	1	1	0	1	0
0	1	1	0	0	1	1	1	0
0	1	1	1	0	1	0	1	0
1	0	0	0	1	1	1	1	1
1	0	0	1	1	1	0	1	1
1	0	1	0	0	0	1	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	1	1	1	1	1
1	1	0	1	1	1	0	1	1
1	1	1	0	0	1	1	1	1
1	1	1	1	0	1	0	1	1

c) A.B.C + A . notB . notC + notA . notB. notC

A	B	C	A.B.C	notA.notB .notC	notB . notC	A. notB . notC	A.B.C + A . notB . notC + notA . notB. NotC
0	0	0	0	1	1	0	1
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	0	0	1	1	1
1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	1	0	0	0	1

d) (A+B). (not(A + C)) . (not (notA xor B))

A	B	C	(A+B)	(A+C)	not(A + C)	NotA	NotA xor B	not (notA xor B)	(A+B). (not (A + C)) . (not (notA xor B))
0	0	0	0	0	1	1	1	0	0
0	0	1	0	1	0	1	1	0	0
0	1	0	1	0	1	1	0	1	1
0	1	1	1	1	0	1	0	1	0
1	0	0	1	1	0	0	0	1	0
1	0	1	1	1	0	0	0	1	0
1	1	0	1	1	0	0	1	0	0
1	1	1	1	1	0	0	1	0	0

e) A.B + A. notB

A	B	A.B	notB	A. notB	A.B + A. notB
0	0	0	1	0	0
0	1	0	0	0	0
1	0	0	1	1	1
1	1	1	0	0	1

f) A+ Not(notB + A.C) xor (notD)

A	B	C	D	A.C	notB	notB + A.C	Not(not B + A.C)	A+ Not(notB + A.C)	not D	A+ Not(notB + A.C) xor notD
0	0	0	0	0	1	1	0	0	1	1
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	0	1	1	0	0	1	1
0	0	1	1	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1	1	1	0
0	1	0	1	0	0	0	1	1	0	1
0	1	1	0	0	0	0	1	1	1	0
0	1	1	1	0	0	0	1	1	0	1
1	0	0	0	0	1	1	0	1	1	0
1	0	0	1	0	1	1	0	1	0	1
1	0	1	0	1	1	1	0	1	1	0
1	0	1	1	1	1	1	0	1	0	1
1	1	0	0	0	0	0	1	1	1	0
1	1	0	1	0	0	0	1	1	0	1
1	1	1	0	1	0	1	0	1	1	0
1	1	1	1	1	0	1	0	1	0	1

2) a) A. notB + E.D
   b) A.B.C
   c) A.(B + C)
   d) X.Y.Z.R.S.T
   e) A
   f) A . C

3) a) X = 1001
   b) X = 0000
   c) X = 1001
   d) X = 0000
   e) X = 1101

4)

X	Y	X + Y	(X+Y) . Y	(X.Y)+Y	(X.Y)
0	0	0	**0**	**0**	0
0	0	0	**0**	**0**	0
0	1	1	**1**	**1**	0
0	1	1	**1**	**1**	0
1	0	1	**0**	**0**	0
1	0	1	**0**	**0**	0
1	1	1	**1**	**1**	1
1	1	1	**1**	**1**	1

5) X = not (A.B.C.D)

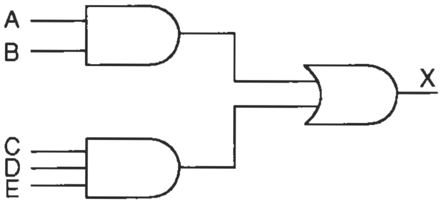
6) a) XY   b) X   c) X (Z + Y)   d) 1   e) A + C

7) Simplificando, F = A + B + C
   Utilizando uma NAND: F = not (notA . notB . notC)

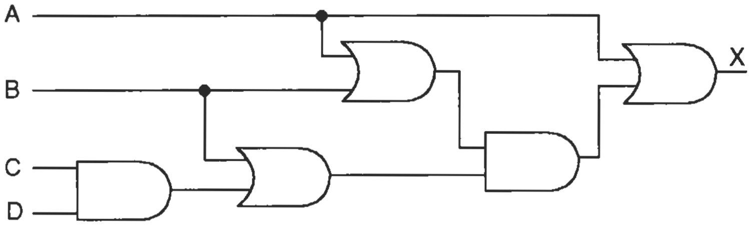
8) São portas construídas por técnica especial, na qual várias portas AND ou OR são conectadas diretamente através de portas NAND ou NOR.

9) Têm desempenho (velocidade) superior, a custo superior.

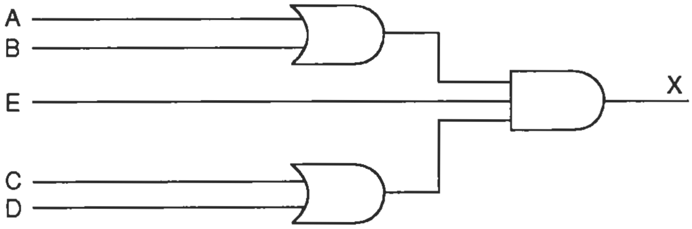
10) a)



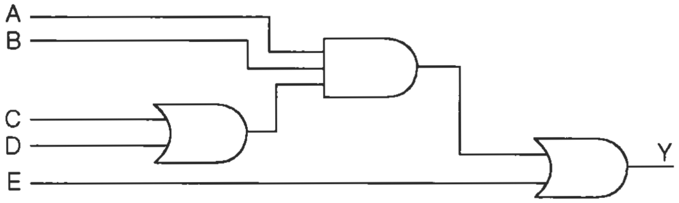
b)



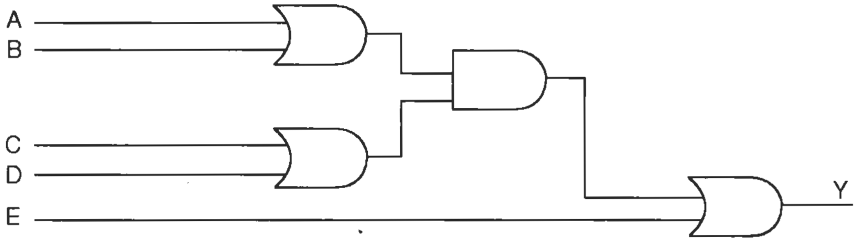
c)



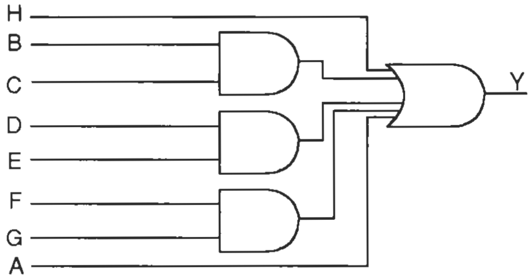
d)



e)



f)



11) A = (F . P) + T
Em que A = alarme; F = sinal de falha; P = sinal de parada e T = sinal de alerta.

12) C = E xor A
Em que C = computador; E = sinal de energia e A = sinal de força alternativa.

13) B = Bruno vai ao jogo
F = Felipe vai à praia
R_a = Renata traz o livro
P = Patrícia traz o livro
R_e = Roberta vai estudar
B = F . (R_e + P)
Como Felipe não quer ir à praia, então Bruno não irá ao jogo.

CAPÍTULO 5

1) Capacidade em bytes = 4098 * 2 = 8.196 bytes

2) O acesso à informação (conteúdo) armazenada na memória (em um endereço). Já o *tempo de acesso* é o período de tempo decorrido desde o instante em que foi iniciada a operação de acesso (endereço é colocado na barra de endereços) até que a informação requerida tenha sido efetivamente transferida.

3) Leitura e escrita

4) SRAM: memória estática, em que o valor de um bit permanece armazenado enquanto houver energia elétrica.
DRAM: memória dinâmica, em que é necessário um circuito de *refresh*, reconstituindo repetidamente o valor de cada bit armazenado.

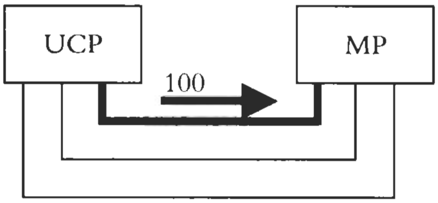
5)

Memória	Endereço	Conteúdo	Total de bits
A	15 bits	8	32K * 8 = **256K bits**
B	14 bits	16	16K * 16 = **256K bits**
C	14 bits	8	16K * 8 = **128K bits**

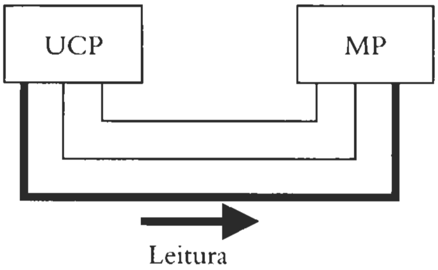
6) REM: armazenar temporariamente o endereço de acesso a uma posição de memória, ao se iniciar uma operação de leitura ou escrita.
RDM: armazenar temporariamente uma informação que esteja sendo transferida da memória principal para a UCP (leitura) ou vice-versa (escrita).

7) Barramento (ou barra) de endereços: interliga UCP à MP, transferindo bits que significam um endereço. É unidirecional, ou seja, a informação trafega da UCP para a MP.
Barramento (ou barra) de dados: interliga UCP à MP, transferindo bits de informação. É bidirecional, isto é, os bits percorrem o barramento da UCP para a MP (operação de *escrita*) e no sentido inverso (operação de *leitura*).
Barramento (ou barra) de controle: interliga UCP à MP para a passagem de sinais de controle (leitura e escrita).

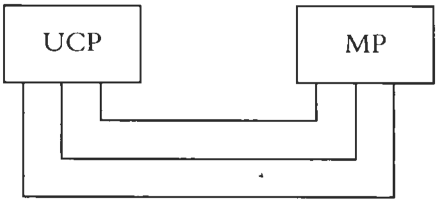
8) I. Endereço (p. ex., 100) é enviado na barra de endereços



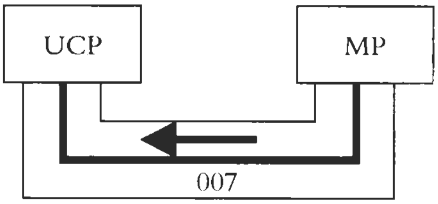
II. Sinal de leitura na barra de controle



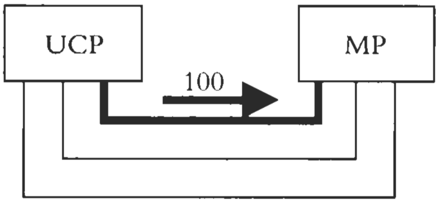
III. MP "acorda" a célula



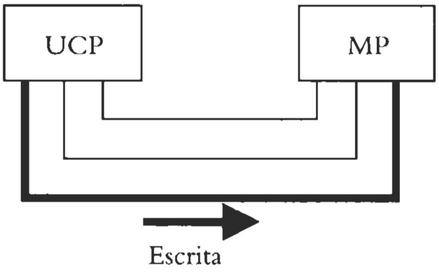
IV. MP envia informação pela barra de dados



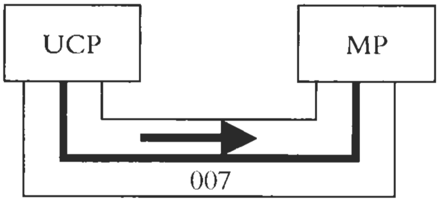
9) I. Endereço (p. ex., 100) é colocado na barra de endereços



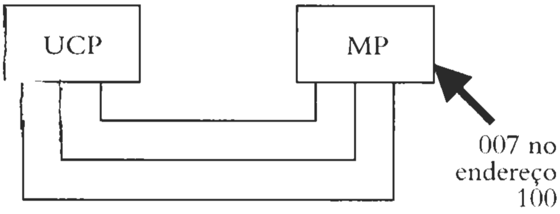
II. Sinal de escrita na barra de controle



III. Informação flui na barra de dados



IV. Memória escreve no endereço 100



10) a) REM = 20 → **E = 20 bits.**
b) Célula = 8 bits; RDM = barra de dados = 16 bits → **2 células.**
c) N = 2^{20} = 1M endereços; cada endereço tem 8 bits → 1M * 8 = **8M bits.**

11) a) N = 32K células = 2^{15}; barra de endereços de 15 bits → maior endereço = 111111111111111 = **7FFF h.**
b) 15 bits.
c) REM = barra de endereços → 15 bits; RDM = palavra = barra de dados → **8 bits.**
d) N = 2^{15}; M = 8; total de bits = N * M → 2^{15} * 8 = 2^{18} = **256K bits.**

12) a) 2C81 = 0010 1011 1000 0001 → REM = 16 bits; F5A = 1111 0101 1010 → RDM = 12 bits.
b) N * M ; N = 2^{16}; M = 12 → 2^{16} * 12 = 3* 2^{18} = **758K bits.**

13) Endereço inicial do 1.° "A" + 128 = endereço do 1.° "B" (segunda letra)
Endereço inicial do 1.° "A" + 2 * 128 = endereço do 1.° "C" (terceira letra)
Endereço inicial do 1.° "A" + 9 * 128 = endereço do 1.° "J" (décima letra)

9 * 128 = 1152 = 480 h
27FA h + 480 h = **2C7A h**

14) Memórias RAM são, tipicamente, muito mais extensas do que as memórias ROM. Caso utilizassem a organização do tipo linear, necessitariam de decodificadores (e, conseqüentemente, de circuitos eletrônicos) muito complexos. Essa desvantagem tem preponderado sobre a vantagem do melhor desempenho desse tipo de organização.

15) Tamanho da MP: 4K * 128 = 512K palavras = 2^{19} → **Endereços de 19 bits**
Número de palavras por quadro: 128 → **campo endereço da palavra = 7 bits**
Número de conjuntos = 64 → **campo n.° do conjunto = 6 bits**
19 – 7 – 6 = 6 → **campo TAG = 6 bits**

TAG	N.° Conjunto	End. Palavra
6	6	7

16) 1.

17) DRAM SRAM

$$\begin{aligned} 5{,}00 + 0{,}00001x &= 1{,}00 + 0{,}00002x \\ (0{,}00002 - 0{,}00001)x &= 5{,}00 - 1{,}00 \\ 0{,}00001x &= 4{,}00 \\ \mathbf{x} &= \textbf{40.000 bits (aproximadamente 40K bits)} \end{aligned}$$

18) N = 1M = 2^{20}
Barra de endereços = REM = 20 bits → **20 flip-flops, um para cada bit**
M = 8 bits
Assumindo a palavra = célula:
Barra de dados = RDM = 8 bits → **8 flip-flops, um para cada bit**

19) Bipolar e MOS (Metal Oxide Semiconductor)

20)

	Memória Principal	**Memória Cache**
Tempo de Acesso	50–10 nanossegundos	5–10 nanossegundos
Capacidade	Alta: tipicamente 64 Mbytes final (década 90)	Média
Temporariedade	Média: dados permanecem mais tempo do que na cache	Pequena: menor que a duração da execução dos programas

21) Quando o método de mapeamento dos quadros de cada conjunto é a *escolha aleatória*.

22) SIMM — Single in line Memory Modele — trata-se de um módulo ("pente") contendo mais de uma pastilha de memória e com um sistema de encaixe em slots, semelhante ao dos processadores.
EDO DRAM é um tipo de memória DRAM, mais rápida que as DRAM convencionais.
Não, são tecnologias diferentes.

23) Não, as memórias ROM permitem apenas leitura, diferentemente das de leitura/escrita. O próprio nome (Read Only Memory) já indica sua propriedade.

24) a) Tag = 8 bits Linha = 5 bits Byte = 3 bits
b) 9
c) 256 bytes

25) A diferença básica entre memória tipo PROM e EPROM é a capacidade de reutilização que as do tipo EPROM possuem (podem ser apagadas e se reescrever dados) e que as PROM não possuem (só podem ser escritas uma única vez).

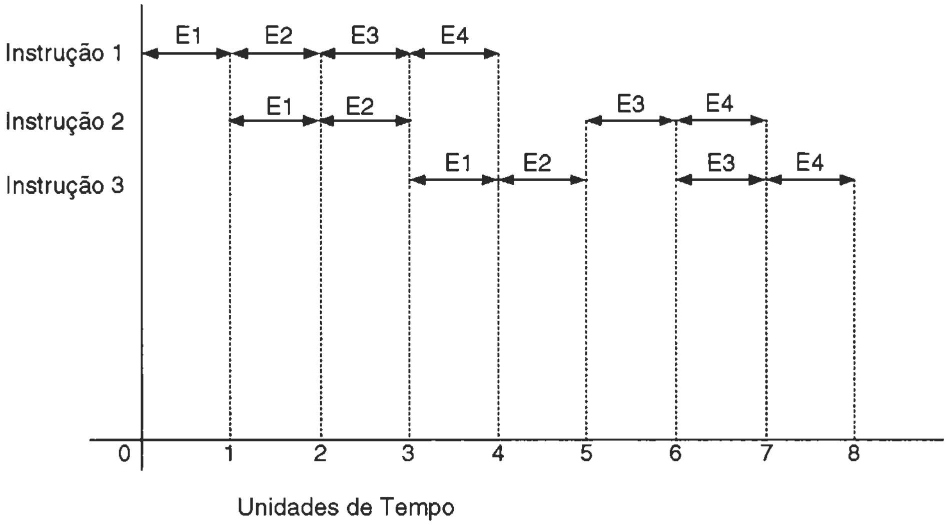
26) A memória ROM original tem seus dados gravados durante o processo de fabricação (como os processadores), enquanto as memórias PROM são construídas sem dados (virgens), que são gravados posteriormente por equipamento especial.

27) Consiste em se utilizar parte da RAM para armazenar o conteúdo da ROM, de modo que o acesso do processador à ROM se faça efetivamente pela RAM, sendo mais rápido (as memórias RAM são muito mais rápidas que as ROM).

28) a) Registradores
b) Memória cache L1 } SRAM
c) Memória cache L2 }
d) Memória Principal DRAM
e) Memória Secundária — Discos magnéticos
Fitas magnéticas
CD-ROM
CD-R
CD-R/W
Disquetes

CAPÍTULO 6

1)



Estágio 1- E1
Estágio 2- E2
Estágio 3- E3
Estágio 4- E4

2) Função Processamento: associada às atividades de efetiva execução de operações matemáticas, lógicas e outras. Seus componentes principais são: a Unidade Aritmética e Lógica (UAL) e os registradores de uso geral.

Função Controle: associada às atividades de busca, interpretação e controle da execução das instruções, bem como o controle dos demais componentes do sistema de computação. Seus principais componentes são: a Unidade de Controle (UC), o Decodificador, o Registrador de Instruções (RI), o Contador de Instruções (CI), o Relógio, o Registrador de Endereço de Memória (REM) e o Registrador de Dados de Memória (RDM).

3) Executar as operações com os dados. Essas operações podem ser:
Aritméticas — soma, subtração, multiplicação, divisão
Lógicas — AND, OR, XOR, NOT
Outras — complementos, deslocamentos à esquerda e à direita

4) É um registrador de uso geral que tem uma função adicional, existente em alguns sistemas: fazer a ligação da UAL com os demais dispositivos da UCP.

5) Relógio

6) Realizar a movimentação de dados de e para a UCP e controlar a ação da Unidade Aritmética e Lógica.

7) Não, seguindo o modelo de von Neumann, em que instruções e dados ocupam a mesma memória. Ambos os registradores armazenam endereços e devem ter o mesmo tamanho.

8) C. Op. = 8 bits: $2^8 = 256$ instruções diferentes

- C. Op. entre 0 e 84: instruções de 16 bits de tamanho (CI ← CI + 2) após a busca da instrução
- C. Op. entre 85 e 170: instruções de 32 bits de tamanho (CI ← CI + 4) após a busca da instrução
- C. Op. entre 171 e 255: instruções de 48 bits de tamanho (CI ← CI + 6) após a busca da instrução

9) Através da inspeção do campo "Código de Operação". Caso ele não seja um desvio, fazer a busca no endereço seguinte (ou em N endereços seguintes, dependendo do valor do campo). Caso contrário, utilizar o campo "Operando" como endereço da próxima instrução.

10) CI — Contador de Instruções (seria melhor chamado *Apontador de Instruções*). Fica na UCP.

11)

ADD Op.	ACC ← ACC +(Op.)	1L + 1L = 2 ciclos
SUB Op.	ACC ← ACC - (Op.)	1L + 1L = 2 ciclos
ADD Op.1,Op.2	(Op.1) ← (Op.1) + (Op.2)	1L + 2L + 1E = 4 ciclos
INCR	ACC ← ACC + 1	1L = 1 ciclo
LDA Op.	ACC = (Op.)	1L + 1L = 2 ciclos
		Total: 2 + 2 + 4 + 1 + 2 = 11 ciclos

12) REM — Registrador de Endereços de Memória. Está ligado diretamente à barra de endereços que, por sua vez, está ligada à Memória Principal.
Se E = número de bits do REM, então $N = 2^E$ (N é a capacidade de memória).

13)

C. Op.	Operando

256 instruções → campo C. Op. = 8 bits ($2^8 = 256$)
64K células → 64K = 2^{16} → endereços de 16 bits
Operando é endereço → operando = 16 bits
Tamanho da instrução: 8 + 16 = 24 bits

Se tamanho da célula = tamanho da palavra = tamanho da instrução
Palavra = RDM = 24 bits
Endereços = CI = 16 bits
ACC acolhe uma palavra = 24 bits
Total de bits = N * M (onde M = bits de cada célula) = 64K * 24 = 1,5M bit

14) a) Barra de dados = 20 bits → RDM = 20 bits; REM = 16 bits → CI = 16 bits.
b) $N = 2^E = 2^{16}$ = 64K células (máxima capacidade). Como a placa tem apenas 4K, a memória poderia ser aumentada em 60K.

c) Se tamanho da célula = tamanho da palavra = tamanho da instrução, então instruções têm 20 bits, com campo "operando" de 16 bits. Campo "C. Op." tem 20 – 16 = 4 bits → com 4 bits, $2^4 = 16$ instruções diferentes.

15) 128 instruções = 2^7 → campo C. Op. = 7 bits; N = 512 = 2^9 → barra de endereços tem 9 bits

$$\underbrace{\underset{7}{\boxed{\text{C. Op.}}}\underset{9}{\boxed{\text{Operando}}}}_{16}$$

a) REM = CI = **9 bits**; RDM = ACC = RI = **16 bits**
b) N = 512 células de 2 bytes (16 bits) cada → N = 512 * 2 = 1024 = **1 Kbyte**
c) REM inalterado (9 bits) → campo "C. Op." teria 8 bits (8 + 9 = 17); $2^8 = 256$ → 256 − 128 = **128 novas instruções**

16) Na verdade estamos nos referindo à aritmética interna do processador. No caso, os registradores e a UAL do sistema A trabalharão, normalmente, com números de 16 bits, enquanto aqueles do sistema B trabalharão com números de 8 bits.
Por exemplo, os microprocessadores Intel Pentium 80486 e 80386 são chips de 32 bits, enquanto os Intel 8086, 8088 e 80286 eram chips de 16 bits. Chips ditos "de 64 bits" são encontrados em estações de trabalho (por exemplo, o ALPHA da Digital/Compac) e suas UAL trabalham efetivamente com números dessa grandeza.
O conceito *palavra* está associado à transferência de informações entre UCP e MP (e vice-versa). Os Pentium têm palavras de 64 bits, enquanto os 80486, 8086 e 8088 têm, respectivamente, 32, 16 e 8 bits de palavra. Em todos os exemplos o tamanho das células é de 8 bits (como padronizado inicialmente pela IBM).

17)

$$38-(16+16)=6\nearrow\quad\underbrace{\boxed{\text{C. Op.}}\underset{16}{\boxed{\text{Op.1}}}\underset{16}{\boxed{\text{Op.2}}}}_{38}$$

a) Se RI = 38 bits → tamanho da instrução = 38 bits
b) 38 − (16 + 16) = 6 bits
c) N = 2^{16} = 64K endereços; se a máquina tem 16K apenas, pode-se acrescentar 48K células (de 38 bits).

18) a) N = 256; M = 16 bits (ou 4 dígitos hexadecimais)
Total de bits na memória = N * M = 256 * 16 = 4.096 = 4K bits.
b) CB05
c) 4040

19) a) t1: REM ← (RI)
Sinal READ
t2: RDM ← (M(REM))
t3: ULA ← ACC
t4: ACC ← RDM
t5: ULA ← ACC
t6: ULA soma
t7: ACC ← ULA

b) t1: CI ← (RI)

c) t1: REM ← (RI)
Sinal READ
t2: RDM ← (M(REM))
t3: ACC ← RDM
t4: ULA ← ACC
t5: ULA ← 1
t7: ULA soma
t8: ACC ← ULA

20) Somadores parciais não possuem entrada para os "vai 1" que porventura ocorram. Apenas possuem entrada para os dois bits a serem somados. Uma outra soma deve ser efetuada a fim de considerar os "vai 1", tornando mais lento o processo. Números com vários algarismos amplificam esse tipo de problema.

21) A 1100 A0 = 0, A1 = 0, A2 = 1, A3 = 1
B 0111 B0 = 1, B1 = 1, B2 = 1, B3 = 0

A soma de A0 = 0 com B0 = 1 não inclui a entrada VUe0 , relativa ao "vai 1". Esta soma resulta no valor 1, sem a geração do "vai 1". No segundo estágio, A1 = 0 e B1 = 1 são somados, sem a entrada de VUe1, uma vez que não houve "vai 1". O resultado é o valor 1, sem a geração de "vai 1". No terceiro estágio, A2 = 1 e B2 = 1 são somados, sem a entrada de VUe2, uma vez que não houve "vai 1". O resultado é o valor 0, e a geração do "vai 1". Este "vai 1" (VUR2) é transferido para o quarto estágio e serve de entrada para VUe3, juntamente com A3 = 1 e B3 = 0. O resultado é o valor 0, com a geração de "vai 1". Em conseqüência, a linha VUR3 indica que houve estouro de algarismos (*overflow*) e o resultado está, assim, incorreto.

22) A 1100 A0 = 0, A1 = 0, A2 = 1, A3 = 1
B 1110 B0 = 0, B1 = 1, B2 = 1, B3 = 1

A soma de A = 0 com B0 = 0 não inclui a entrada VUe0 , relativa ao "vai 1". Esta soma resulta no valor 0, sem a geração do "vai 1". No segundo estágio, A1 = 0 e B1 = 1 são somados, sem a entrada de VUe1, uma vez que não houve "vai 1". O resultado é o valor 1, sem a geração de "vai 1". No terceiro estágio, A2 = 1 e B2 = 1 são somados, sem entrada de VUe1, resultando no valor 0 e na geração de "vai 1". Este "vai 1" (VUR2) é transferido para o quarto estágio e serve como entrada para VUe3, juntamente com A3 = 1 e B3 = 1. O resultado é o valor 1, com a geração de "vai 1". Em conseqüência, a linha VUR3 indica que houve estouro de algarismos (*overflow*) e o resultado está, assim, incorreto.

23) Horizontal: cada bit de microinstrução tem função específica, acessando diretamente uma barra de controle. O formato é simples e direto, mas tende a produzir microinstruções longas demais.
Vertical: em vez dos bits acessarem diretamente uma barra de controle, esses bits podem significar um código de um grupo de ações. A microinstrução é menor, necessitando, entretanto, de um decodificador, tornando o processo um pouco mais lento.

24) Ciclo T1 (no barramento): o ciclo de leitura começa, no início de um pulso de relógio, pela colocação do endereço de leitura no barramento de endereços, emissão de um sinal de leitura (linha READ alta) e elevação da linha de início.
Ciclo T2: a memória decodifica o endereço, coloca os dados no barramento e emite sinal de confirmação.
Ciclo T3: UCP transfere os dados do barramento para o RDM.

Caso a memória não consiga enviar os dados durante o ciclo T2, ela envia sinal "Wait" (estado de espera).

25) O problema reside no fato de que, num desvio condicional, só é possível saber qual a próxima instrução a ser executada durante a execução do desvio. Isso "travaria" a busca antecipada. Existem dois tipos de solução:
1. Buscar as duas opções do desvio (condição verdadeira e falsa).
2. Manter estatística interna à UCP a fim de prever, com uma probabilidade p, a condição (verdadeira ou falsa) e buscar a instrução respectiva.

CAPÍTULO 7

1) a) S&M: 2^{k-1}
b) C1: 2^{k-1}
c) C2: 2^{k-1}

2) A adoção, em larga escala, pelo padrão ASCII, nos anos iniciais da era dos microcomputadores. Isso inibe iniciativas de utilização de outros padrões. Na verdade, a desvantagem dos 7 bits já foi superada pela extensão do padrão aos 8 bits.

3) Trata-se de um código de representação interna, onde cada símbolo é representado por um valor de 16 bits, que pretende englobar todos os símbolos requeridos para as diversas linguagens existentes no planeta. Sua constituição completa e descrição detalhada encontram-se no site do consórcio desenvolvedor: www.unicode.org.

4) a − d = a + C2 (d), onde C2 é o complemento de d (ou o inverso)
C2(d) = 001101 (inverte e soma 1)

$$\begin{array}{r}
11 \\
111000 \\
\underline{001101} + \\
000101
\end{array}$$

b − e = b + C2(e)
C2(e) = 010010

$$\begin{array}{rl} 11001100 & \\ 010010 & + \\ \hline 11011110 & \end{array}$$

c − f = c + C2(f)
C2(f) = 001100001101

$$\begin{array}{rl} 1111 & \\ 001100001101 & \\ 111100001111 & + \\ \hline 001000011100 & \end{array}$$

5) a) 01110111
   b) 1000000001001101
   c) não é possível com 8 bits
   d) 0000000011011001
   e) 111101110010
   f) 1111111100011101

6) a) $-(2^{15} - 1)$ a $+(2^{15} - 1)$
   b) $-(2^{15} - 1)$ a $+(2^{15} - 1)$
   c) -2^{15} a $+(2^{15} - 1)$

7) $$\begin{aligned} C2(C2(N)) &= 2^N - (2^N - N) \\ &= 2^N - 2^N + N \\ &= N \end{aligned}$$

8) O bit de sinal pertence ao número, na aritmética de C2. A magnitude é avaliada por meio de todos os bits, incluindo o de sinal. Ainda assim, todos os números negativos começam com 1. Além disso, não há duplicidade de representação para o zero. Essa é uma vantagem do C2 em relação ao C1 (que também tem duas representações para o zero).

9) O campo representativo da precisão é o da mantissa ou fração, que representa os algarismos significativos do número.

10) a) C1 → $-(2^{15} - 1)$ a 1 $(2^{15} - 1)$
    b) C2 → -2^{15} a $+(2^{15} - 1)$
    c) S/M → $-(2^{11} - 1)$ a $+(2^{11} - 1)$
    d) C2 → -2^{11} a $+(2^{11} - 1)$

11)

a) 0000000000001100	b) 0001101100101010	c) 1111100111111001
d) 0110111101000001	e) 1111111010111000	f) 1000000000000000
g) 1101110111011101	h) 1000000000000001	

12) a) $+0{,}00565 = +0{,}00000001011110010 \star 2^0 = +0{,}1011110010 \star 2^{-7}$

0	10111	1011110010

b) $-674{,}25 = -101010001{,}01 = -101010001{,}01 \star 2^0 = -0{,}101010001 \star 2^9$

1	01001	1010100010

c) $+46{,}5 = +101110{,}1 = +101110{,}1 \star 2^0 = +0{,}1011101 \star 2^6$

0	00110	1011101000

d)

1	10101	1100100010

e)

0	00111	1111110010

13) a) E745 = 1110 0111 0100 0101

1	11001	1101000101

$-0{,}1101000101 * 2^{-9} =$

b) 3FC6 = 0011 1111 1100 0110

0	01111	111100110

$+0{,}111100110 * 2^{+15} = +111100110000000 * 2^{0} = +\ 63.872$

c) F320 = 1111 0011 0010 0000
$-0{,}1100100000 * 2^{-12}$

1	11100	1100100000

14) Padrão IEEE 754: Sn = Sinal do número; E = Expoente (bias); F = Significando (fração ou mantissa).
Formato: Sn E F, sendo: Sn = 1 bit + E = 8 bits + F = 23 bits
E (bias) expresso em excesso de 127 ou E = Valor real do expoente mais 127

a) 0 01110110 01001100101010101110001
b) 1 10001101 10001011100010000000000
c) 1 01111011 00100100010010100110001

15) a) 0 0111110 10111000000010000000000
b) 0 1110000 11011111000000000000000
c) 1 0001110 10000000100000000000000

16) a) 1000110 (ocorreu *overflow*) b) 0100011 c) 1011000
d) 1000010 e) 0110101 (ocorreu *overflow*) f) 1100010

17) Maior valor positivo:

0	0111111	111111111111111111111111

$+0{,}111111111111111111111111 * 2^{+15} =$
$= +111111111111111{,}111111111_2$

Menor valor positivo:

0	1111111	100000000000000000000000

$+0{,}100000000000000000000000 * 2^{-15} = +0{,}1 * 2^{-15}$
$= +0{,}0000000000000001_2$

18) Aumentar o campo destinado à mantissa normalizada.

19) $-(2^{16-1} - 1) = -(2^{15} - 1) = -(32768 - 1) = -32767.$

20) Porque em complemento a 2 aproveita-se a representação 100000...... (número de bits dependente da aritmética do processador) que, em sinal e magnitude ou complemento a 1, não era utilizada, servindo apenas como uma segunda (e indesejável) representação do zero.

21) a) 000000
b) 100100
c) 100000
d) 000000
e) 110011

22) XOR
AND

23) a) 1111110110111000 b) 1111101001010000
c) 111111100100110 d) 1111111010111100

24) a) C1, C2 : 0000000011011011
b) C1: 1111110011111001 C2: 1111110011111010
c) C1: 1111111100011110 C2: 1111111100011111

d) C1, C2: 0000000001110101

25) a) Resp. 11000011

b) Resp. 11 resto 1

c) Resp. 11111100

d) Resp. 100

26) a) 00010101 b) 10011010 c) 00011100 d) 10110110

27) a)

1	01000	1010110100

b)

1	01000	1101101100

c)

0	01000	1110110100

d)

1	01001	1001111100

CAPÍTULO 8

1) A economia de espaço em memória. Instruções com muitos operandos tendem a ocupar muito espaço e demorar mais para serem completamente transferidas para a UCP.

2)
```
ADD Op.1,Op.2    (Op.1) ← (Op.1) + (Op.2)
SUB Op.1,Op.2    (Op.1) ← (Op.1) − (Op.2)
MPY Op.1,Op.2    (Op.1) ← (Op.1) * (Op.2)
DIV Op.1,Op.2    (Op.1) ← (Op.1) / (Op.2)
MOV Op.1,Op.2    (Op.1) ← (Op.2)
```

Supondo que as variáveis foram lidas e encontram-se na MP:

a)
```
SUB C,A
MPY C,B     ; B*(C − A)
DIV E,B     ; E/B
MOV X,D     ; salva D em X
SUB D,E     ; D − E/B
MPY X,D     ; (D − E/B)*D
ADD X,C
ADD X,A
```

b)
```
MOV Y,B     ; salva B em Y
SUB B,F     ; B − F
MOV F,E     ; salva E em F
DIV E,B     ; E/(B − F)
MPY D,E     ; D*E/(B − F))
ADD D,Y     ; D*(E/(B − F)) + B
SUB C,Y     ; C − D*(E/(B − F) + B
MPY Y,C     ; B*(C − D*(E/(B − F)) + B)
MPY Y,F     ; B*(C − D*(E/(B − F)) + B)*E
ADD Y,A
```

3)
```
ADD Op.         ACC ← ACC + (Op.)
SUB Op.         ACC ← ACC − (Op.2)
MPY Op.         ACC ← ACC * (Op.)
DIV Op.         ACC ← ACC / (Op.)
LDA Op.         ACC ← (Op.)
STA Op.         (Op.) ← ACC
```

Supondo que as variáveis foram lidas e encontram-se na MP:

a)
```
LDA A
SUB C
MPY B
STA X    ; B*(C − A)
LDA E
DIV B    ; E/B
STA E
LDA D
SUB E
MPY D    ; (D − E/B)*D
```

b)
```
LDA B
SUB F
STA Y      ; B − F
LDA E
DIV Y
MPY D      ; D*(E/(B − F))
ADD B
STA Y
LDA C
SUB Y      ; (C − D*(E/(B − F)) + B)
```

```
ADD X     ; B*(C - A) + (D - E/B)*D
ADD A
STA X
```

```
MPY E
MPYB
ADD A
STA Y
```

4)
```
ADD Op.1,Op.2,Op.3       (Op.3) ← (Op.1) + (Op.2)
SUB Op.1,Op.2,Op.3       (Op.3) ← (Op.1) - (Op.2)
MPY Op.1,Op.2,Op.3       (Op.3) ← (Op.1) * (Op.2)
DIV Op.1,Op.2,Op.3       (Op.3) ← (Op.1) / (Op.2)
```

Supondo que as variáveis foram lidas e encontram-se na MP:

a)
```
SUB C,A,X
MPY B,C,X     ; B*(C - A)
DIV E,B,E     ; E/B
SUB D,E,E     ; D - E/B
MPY D,E,E     ; (D - E/B)*D
ADD E,X,X
ADD X,A,X
```

b)
```
SUB B,F,F     ; B - F em F
DIV E,F,F     ; E/(B - F) em F
MPY D,F,D     ; D*E/(B - F)) em D
ADD D,B,Y     ; D*(E/(B - F)) + B em Y
SUB C,Y,Y     ; C - D*(E/(B - F) + B em Y
MPY B,Y,Y     ; B*(C - D*(E/(B - F)) + B)
MPY Y,E,Y     ; B*(C - D*(E/(B - F)) + B)*E
ADD Y,A,Y
```

5) A inicialização de contadores. Uma desvantagem é a limitação do tamanho do campo, que reduz o valor máximo do dado a ser manipulado.

6) O acesso às variáveis em memória. É um modo de endereçamento universal. Uma possível desvantagem seria a limitação do número de endereços, pelo limite físico do campo "operando". Entretanto, essa desvantagem pode ser contornada pela combinação do valor (agora chamado **deslocamento**) com o existente em outro registrador (chamado **base**).

7) Ambos obtêm o dado na MP. No caso do modo direto por registrador, a instrução que faz referência ao dado é mais curta, porque contém apenas o endereço do registrador (endereço "curto") que, efetivamente, armazena o endereço do dado.

O modo direto, embora pressuponha instruções mais longas, pode transferir o dado diretamente da memória para a UAL, sem passos intermediários (como colocar o endereço no registrador e aí requerer a sua movimentação para a UCP). A manutenção de informação nos registradores é crítica pelo número reduzido deles na UCP (em máquinas RISC encontram-se, normalmente, muitos registradores, para melhor aproveitamento desse modo).

Assim, a utilização do modo direto por registrador é útil quando se pode manter um endereço em registrador por longo tempo e utilizá-lo repetidamente, dentro de uma iteração. A escolha dessa opção, entretanto, nem sempre é trivial.

8) $X = (A + C + (B * D - E)) / F$

9) Simplificar o acesso aos dados em memória. Cada endereço é formado pela combinação de dois valores (base e deslocamento), ambos de tamanho menor que um endereço completo. A base pode permanecer num registrador e ser referenciada na instrução pela indicação do número do registrador. O deslocamento é referenciado, normalmente, na própria instrução.

Não há diferença de implementação entre o modo base mais deslocamento e modo indexado. A diferença está apenas na aplicação: enquanto o primeiro normalmente está ligado à obtenção de dados simples/instruções na MP, o segundo otimiza o endereçamento de estruturas de dados mais complexas, como vetores, por exemplo.

10) 2^C. Se C bits formam o campo deslocamento, então se pode gerar 2^C números diferentes, que seriam os endereços acessáveis com um único valor de registrador-base.

11) R_A = registrador de 16 bits
R_B = registrador de 8 bits
Op. = operando de 16 bits

Instrução	Descrição	Modo de Endereçamento
LDA Op.	$R_A \leftarrow$ (Op.)	modo direto
ADD Op.	$R_A \leftarrow R_A$ + Op.	modo direto

LDB Op.	$R_A \leftarrow ((R_B) + Op.)$	modo indexado
LDR Op.	$R_B \leftarrow Op.$	modo imediato

Programa

```
LDR  013D     ; carrega endereço do início da tabela
LDB  1F       ; acessa o elemento, no modo indexado
```

12) a) Campos R1 e R2 = 24 − 12 − 8 = 4
R1 = 2 bits ; R2 = 2 bits
Com 2 bits tem-se, no máximo, $2^2 = 4$ **registradores**;

b) Instrução A: Instrução B:

Instrução A: Acessos	Ciclos	Instrução B: Acessos	Ciclos
Busca instrução	1	Busca Instrução	1
Busca endereço	1	Busca dado	1
Busca dado	1		
Total	3	Total	2

No modo registrador indireto, a instrução acessa o valor anteriormente inserido num registrador e busca o conteúdo de memória associado ao endereço que esse valor significa. Já no modo indireto a execução da instrução prevê duas idas à memória. No total (incluindo a busca da instrução) serão 2 acessos no modo-registrador indireto e 3 no modo indireto.

CAPÍTULO 9

1) É o processo de transformação de um programa escrito em linguagem de alto nível (o programa-fonte) em um programa em linguagem de máquina (o programa-objeto). Esse processo é executado pelo programa *compilador.*

2) É um processo de execução de programas que, diferentemente da compilação, não gera um programa-objeto a partir de um programa-fonte. Nesse processo, o programa *interpretador* lê o programa-fonte e o executa diretamente.

3) Linguagens interpretadas facilitam a depuração de programas-fonte, porque os erros podem ser apontados com mais precisão. Além disso, programas-fonte podem ser executados em diferentes SC, desde que estes contenham os interpretadores adequados. É o caso dos "applets", programas "semiprontos", que trafegam pela Internet e são executados em qualquer computador a ela conectado (o interpretador está embutido no programa de navegação). Linguagens compiladas, por outro lado, têm a vantagem de executar mais "rápido" através dos seus programas executáveis. Isso é particularmente notável dentro dos loops onde, no caso da interpretação, o mesmo código é novamente interpretado a cada iteração, com conseqüente sobrecarga da UCP.

4) Porque a Linguagem Assembly ainda não é a Linguagem de Máquina, ainda que muito próxima. Embora a Linguagem Assembly seja fundamentalmente calcada nas características do processador, ela ainda necessita de uma pequena transformação (chamada de Montagem) para ser executável. Essa é a forma de se resolver o "problema": transformar o programa escrito em Linguagem Assembly em Linguagem de Máquina por meio do programa Montador (ou Assembler).

5) Como já afirmado, um compilador transforma Programas-fonte em Programas-objeto. Os primeiros são escritos em linguagens de programação específicas e formais. Os últimos são feitos para executar em determinado processador, porque contêm instruções projetadas para isso. Portanto, um compilador está "amarrado" a ambas as condicionantes.

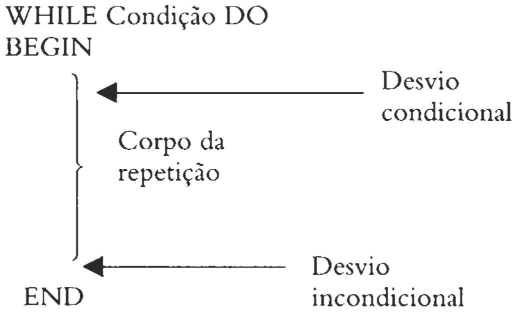
6) O processo de ligação une dois ou mais programas-objeto, tornando-os um único programa executável. Entre os programas-objeto podem estar as *bibliotecas* da Linguagem, que são conjuntos de rotinas pré-compiladas e disponibilizadas pelo fabricante do compilador. Durante a mencionada união são resolvidos os problemas referentes às *referências externas*, ou seja, chamadas ou desvios entre programas-objeto (também chamados de módulos ou unidades) distintos.

7) a) RI = 0000 (última instrução executada); CI = 1C0 (a próxima instrução a ser buscada); ACC = 002B (= 43 decimal).
b) 43.
c) O Código de Operação 8 não seria reconhecido, e o sistema, provavelmente, pararia, necessitando de reinicialização.

8) O código-objeto contém referências externas abertas, isto é, chamadas a rotinas ou programas que não estejam no seu corpo. O módulo de carga (ou programa executável) já tem as referências externas resolvidas, isto é, não há instruções de desvio incompletas. Ambos têm em comum a linguagem de máquina (zeros e uns).

9) Uma chamada (instrução *call*) ou desvio para um endereço que, em tempo de compilação, seja desconhecido, por estar em outro módulo (ou unidade), ainda não compilado.

10) O desvio altera a seqüência de execução de instruções, pela modificação do conteúdo do registrador CI. Desvios podem ser incondicionais ou condicionais. Estes últimos testam certas ocorrências (condições) que, se satisfeitas, confirmam o desvio. Em linguagens de alto nível, os comandos de seleção e repetição implementam desvios (condicionais e incondicionais). Por exemplo, ao final do corpo da repetição "WHILE Condição DO..." de Pascal, há um desvio incondicional para o início do corpo, onde há um desvio condicional: caso a condição seja verdadeira, retorna-se ao corpo da repetição. Caso contrário, desvia-se para a primeira instrução após a repetição.



11) Análises léxica, sintática e semântica.

12) Não. O código gerado contém as instruções que o processador Intel 80486 consegue executar. Tal código é incompreensível para o Macintosh, que possui jogo de instruções completamente diferente.

CAPÍTULO 10

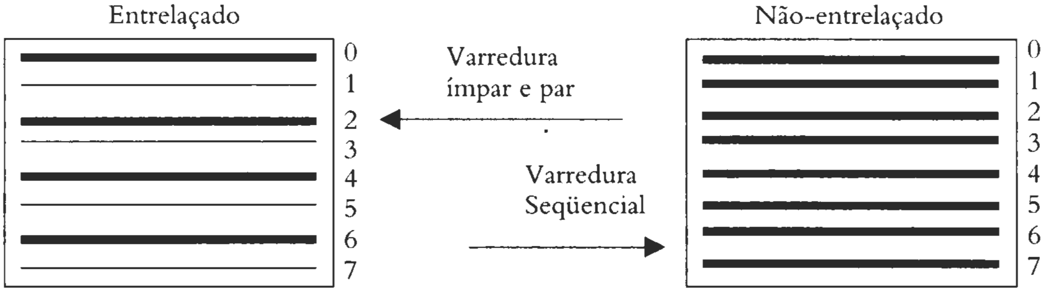
1) Um dispositivo de hardware que compatibiliza um periférico qualquer (que tem características de projeto e fabricação próprias) com o barramento principal (que possui suas próprias características). O interface conecta, assim, o periférico à UCP, funcionando como um intermediário entre esses componentes.

2) É o tempo gasto para interpretação do endereço pela unidade de controle e movimento mecânico do braço, para cima da trilha desejada. É o maior componente do chamado tempo de acesso a disco.

3) Controladoras de disco, "adaptadores" de vídeo, placas de som etc.

4) Serão utilizados 15 cilindros (cada um contém um total de 200 setores). Serão 15 acessos, perfazendo um total de 15 * 23 = 345 ms para busca e latência. A esse total ainda se acrescentará o tempo de transferência dos 20 setores consecutivos de cada trilha acessada.

5)



6) A metade do total de linhas do quadro (uma varredura para as linhas pares e outra para as linhas ímpares).

7) Barramento tipo múltiplo (com barramentos separados de Dados, Endereços e Controle) e barramento tipo único (Unibus).

8) A informação a ser transmitida/recebida bit a bit, um após o outro.

9) Nesse caso, o periférico é conectado ao interface por uma única linha de transmissão de dados (há outras para controle). Assim, um bit é transmitido de cada vez. A construção é mais simples, adequada a periféricos de baixa velocidade, como o teclado ou mouse, por exemplo.

10) A informação a ser transmitida/recebida em grupos de bits de cada vez.

11) Nesse caso, cada bit do grupo é enviado/recebido em uma linha separada. A velocidade de transferência é grande, ideal para periféricos de alto desempenho, como discos rígidos ou modernas impressoras. O custo de fabricação é maior.

12) Intel 8048 ou Intel 8049, ambos de 8 bits.

13) Detecção do pressionamento de tecla, confirmação do pressionamento, geração do código correspondente à tecla, geração da interrupção, transmissão do código para área da memória principal e análise do código enviado pelo Sistema Operacional.

14) Modos de funcionamento diversos, velocidades de transferência diferentes e formatos/tamanhos de unidades de transferência diversos.

15) I. Interface interroga periférico sobre a disponibilidade em receber dados
II. Periférico responde
III. Interface transmite os dados
IV. Periférico certifica recebimento ou término de leitura

16) É baseado em agulhas (9 a 24) dispostas em forma de matriz que, ao serem acionadas eletronicamente por bobinas, impactam sobre uma fita de tinta, marcando o papel. Cada caractere é impresso dessa forma.

17) Num cilindro fotossensitivo é formada a imagem da página a ser impressa. Um toner espalha partículas sobre a imagem no cilindro. A imagem é transferida para o papel, sendo secada por intenso calor logo a seguir. O cilindro é apagado, para uma nova impressão.

18) De forma parecida com as impressoras matriciais. Em vez de agulhas, possuem pequenos tubos com bicos que permitem a saída de gotas de tinta. O maior número de bicos aumenta a densidade de impressão e, conseqüentemente, a sua qualidade. As gotas de tinta são expelidas pelo calor provocado por resistência elétrica próxima aos tubos.

19) FB = NRL/NRL
NRF = 30.000 / 15 = 2000 (número de registros físicos)
Registro físico: (150 bytes * 15) + 150 bytes = 2250 + 150 = 2400 bytes
Capacidade da trilha: 7200 bytes = 3 registros físicos
1 trilha = 3 RF
N trilhas = 2000 RF
N = 666,666 = 667 trilhas

20) a) 1 cilindro = 13.030 * 18 = 234.540 bytes
20 cilindros no bloco = 20 * 234.540 = 4.690.800 bytes disponíveis
Se o tamanho do bloco = 4.335 → 4.690.800 / 4335 = 1082,07 → cabem **1082 blocos (NRF — Registros Físicos) nesse espaço.**
Cada bloco possui 4335 − 135 = 4200 bytes úteis → ou 10 registros lógicos de **420 bytes cada.**
Total de registros lógicos: 1082 * 10 = **10820**
NRL = FB * NRF → NRL = 10 * 1082 = 10820 registros lógicos
FB = RL/RF → RL = 4200 * 10 = 42.000 bytes = tamanho de cada registro lógico

b) Tamanho do bloco (sem informação de controle): 4200 bytes
Quantidade de blocos : 1082
Espaço gasto com gaps: 1082 * 0,75″ = 811,5″
Bytes totais: 1082 * 4200 = 4.544.400 bytes
Espaço necessário para os bytes: 4.544.400 / 1600 = 2.840,25″
Espaço total necessário (bytes + gaps) = 3.651,75″
Em pés: 3.651,75 / 304,32 pés

21) É um método de realização de operações de E/S. A sua importância está no fato de não exigir que a UCP permaneça em contínua atenção às necessidades dos periféricos.

22) De modo geral, consiste na realização de transferência de dados entre um determinado interface e a memória principal, praticamente sem intervenção da UCP. O controlador de DMA é quem efetivamente controla o barramento e os componentes envolvidos, sob solicitação da UCP. A principal vantagem é permitir à UCP a realização de outras tarefas enquanto a transferência está em progresso.

23) Integrando o sistema disco/acionador/atuador, a tecnologia Winchester evitou problemas ligados a desalinhamentos e possibilitou o aumento da densidade de gravação e quantidade de trilhas.

24) Como os braços/cabeças se movimentam juntos, quando o atuador se desloca para acessar uma determinada trilha de certa superfície, todas as cabeças estacionam sobre a trilha de mesmo endereço em todas as superfícies (formando o chamado *cilindro de trilhas*). Armazenando-se um arquivo dessa forma, a sua leitura/gravação é muito mais rápida, porque o movimento dos braços e cabeças é minimizado.

25) Técnica simples (e mais antiga) de transmissão serial de bits, em que um caractere é transmitido/recebido de cada vez, havendo uma ressincronização do transmissor e do receptor após o evento. Para isso, alguns bits especiais foram criados, sendo chamados bits de partida e de parada, demarcando o início e o fim, respectivamente, do caractere.

26) Nessa técnica de transmissão serial de bits são transmitidos blocos maiores de bits a cada vez, e não pequenos grupos de 8 bits, como no caso assíncrono. Para isso é necessário um melhor sincronismo entre transmissor e receptor (por linha dedicada ao pulso de relógio ou mesmo a inclusão desses pulsos dentro da informação). Para cada bloco de caracteres de informação existe um grupo de caracteres de controle.

27) $$\frac{8 \star 200}{(8 + 1 + 1) \star 200} = \frac{1600}{2000} = 80\% \text{ de eficiência}$$

Número total de bits transmitidos: 2000 * 8 = 16.000
Tempo de transmissão = 16.000 / 2000 = 8 segundos.

28) É um dispositivo que decompõe o caractere recebido em bits e retira os bits de partida e parada. Na transmissão, inversamente, ele inclui os mencionados bits e monta o caractere. Isto é necessário porque internamente na UCP e no barramento os bits sempre trafegarão em paralelo. A UART promove a "serialização" dos mesmos. UARTs possuem buffers para receber os bits e registradores especiais que deslocam cada bit de um caractere. Um relógio controla as ações do mecanismo, e uma unidade de controle permite que ela funcione de diversas maneiras (com ou sem paridade etc).

29) O espaço entre superfície e cabeça de gravação/leitura é mínimo. Para esses padrões, a existência de poeira é potencialmente fatal para o funcionamento do disco.

30) Os endereços de interfaces de E/S são específicos, não "competindo" com o espaço de endereçamento da memória principal. Existe, assim, memória e endereços próprios para armazenar instruções/dados de operações de E/S. Esse tipo de organização tem a vantagem de não consumir endereços da MP. Em compensação, exige sinais de controle (ou instruções) especiais, utilizados para que o interface perceba que o endereço que trafega na barra de endereços não é um endereço de MP.

31) Nesse caso, as instruções/dados de programas comuns compartilham a memória com as instruções/dados de operações de E/S. Os sinais de controle ficam simplificados, pois os endereços de E/S estão perfeitamente definidos, ao custo do consumo adicional de memória.

32) Como mencionado nos exercícios anteriores, ambas têm vantagens e desvantagens. A memória isolada poupa a memória principal, mas exige sinais de controle e instruções especiais, que residirão na memória especialmente criada para esse fim (E/S). No caso de memória compartilhada, a mesma memória é utilizada para uso de programas e operações de E/S, simplificando os sinais de controle, mas reduzindo o espaço da MP para os programas comuns.

33) A tecnologia capacitiva. Uma tecla capacitiva funciona na base da variação de capacitância do acoplamento entre duas placas metálicas, variação essa que ocorre quando a tecla é apertada.

34) VRC (válvula de raios catódicos), LED (diodos emissores de luz), LCD (cristal líquido) e FDP (vídeos planos).

35) Dispositivo de entrada cujo propósito é facilitar a comunicação do usuário com o sistema, "apontando" suas opções na tela do monitor de vídeo. Um sensor sob o mouse (mecânico, ótico ou ótico-mecânico) capta o movimento em uma superfície plana e o transmite ao SC. O usuário escolhe o que quer apontar e seleciona, apertando um botão.

36) No primeiro, a tela de vídeo é dividida em linhas e colunas, formando uma matriz em que cada encontro (linha, coluna) é usado para representar um símbolo válido (por exemplo, um caractere ASCII). No segundo, a tela é uma única matriz de pontos, chamados "pixels", que têm os atributos ligado-desligado, cor etc. Um caractere será uma matriz de pixels.

37) Como dito no exercício anterior, um pixel é um ponto da matriz de pontos em que a tela do monitor de vídeo é dividida.

38) Resolução está ligada à quantidade de pixels que se pretende apresentar em tela. Por exemplo, 1024 × 768 pontos é considerada uma alta resolução. Dependendo da variedade de cores que cada ponto possa ter, a necessidade de memória de vídeo para armazenar toda a informação relativa aos pixels será muito grande. Por exemplo, se cada pixel precisar de 16 bits para representar suas cores (16 cores, portanto), então serão necessários 1024 × 768 × 16 bits ou 1.572.864 bytes (quase 2Mb) para armazenar uma única tela cheia.

39) Gap é um espaço morto deliberadamente inserido entre blocos de informação, com o propósito de permitir a aceleração/desaceleração da fita sem haver perda de leitura do início de um novo bloco. Densidade tem a ver com a distância entre bits gravados e entre as colunas (trilhas) de gravação. Quanto maior a densidade, mais informação pode ser armazenada num determinado trecho de fita magnética.

40) O intenso uso da UCP para tarefas de monitoração e controle dos interfaces, prejudicando o funcionamento do SC. A alternativa é a técnica de Interrupção, em que o interface "avisa" ou "solicita atenção" da UCP por meio de sinais elétricos (as interrupções).

41) A interrupção interna ocorre devido a algum evento gerado pela execução de uma instrução ou mesmo programado (por exemplo, uma divisão por zero). Também é chamada de interrupção por software. A interrupção externa (ou interrupção por hardware) está ligada a um interface de E/S que pretende "avisar" à UCP a necessidade de atenção para o seu periférico.

42) É a confirmação do pressionamento de uma tecla. Para isso, o processador embutido no teclado repete várias vezes a varredura sobre a referida tecla.

43) Cristal líquido: a matriz de pixels no monitor é preenchida com cristais especiais, que permanecem em estado intermediário líquido-sólido. Esses cristais são sensíveis à polarização elétrica. Normalmente refletem a luz que incide sobre o painel (vídeo). Entretanto, quando uma voltagem é aplicada sobre o cristal (isso é feito para cada pixel), suas moléculas se modificam, deixam a luz passar para ser absorvida por um polarizador. Essa é uma explicação simples. Os LCD têm evoluído bastante nos últimos anos, diminuindo cada vez mais a dependência da luz externa incidente.

Gás plasma: Um gás (néon) é excitado pela aplicação de voltagem, emitindo luz avermelhada. Uma matriz de eletrodos permite endereçar todos os pixels. Como esse sistema não depende de luz externa, precisa de muita energia para funcionar, o que restringe a aplicação em máquinas portáteis.

44) A imensa maioria dos laptops (ou notebooks) usa essa tecnologia, basicamente em dois grandes ramos: matriz passiva e matriz ativa (esta mais eficaz).

45) Scanners convertem imagens em papel para pontos (pixels), que são codificados em forma binária. Para isso, um feixe luminoso é acionado, percorrendo o papel do início ao fim. A luz é refletida, atingindo um dispositivo chamado CCD, que reage, produzindo um sinal elétrico em cada uma de suas células fotossensitivas (são mais de 2500). Cada sinal elétrico será proporcional à intensidade da luz refletida. Esses sinais se converterão nos pixels da imagem, sendo enviados ao computador sob códigos binários.

46) 5180 / 8,5 = 609,4 → 600 dpi é a resolução.

47) Porque foram projetados para priorizar a capacidade de armazenamento em detrimento da taxa de transferência.

48) a) O armazenamento de dados se dá em uma única trilha em espiral, ideal para grandes blocos de dados (como os CD de áudio). Assim, a localização de pequenos registros é dificultada.
    b) Um CD-ROM armazena com densidade fixa, isto é, com distância entre bits constante. Isso implica a necessidade de acessar os trechos mais externos com velocidades diferentes dos trechos internos (o que não ocorre com os discos rígidos). Essa variação de velocidade reduz a taxa de transferência.

CAPÍTULO 11

1) A evolução das linguagens de alto nível levou ao estabelecimento de comandos mais complexos, de forma a simplificar e facilitar o trabalho de programadores. Entretanto, as instruções de máquina continuavam rudimentares, fazendo com que os compiladores da época fossem obrigados a gerar código complexo e ineficiente para implementar aqueles novos e sofisticados comandos. A essa diferença de evolução entre as linguagens de alto nível e as linguagens de máquina chamou-se "gap semântico".

2) Aumentar o número de instruções, incluir mais modos de endereçamento e utilizar mais microprogramação. Em outras palavras, a sofisticação do hardware, influenciada pela evolução das linguagens de alto nível.

3)

Característica	RISC	CISC
Quantidade de instruções	Pequena execução otimizada	Grande
Execução de chamada de funções	Ocorrem no processador (mais registradores)	É freqüente a utilização de memória para manipulação de parâmetros
Quantidade de modos de endereçamento	Poucos	Muitos
Modo de execução com pipelining	Uso muito intenso	Uso normal

4) Arquitetura RISC de alto desempenho, utilizado no sistema RISC 6000 e Macintosh.

5) A otimização da arquitetura pela divisão funcional de tarefas (processador de desvio, processador de inteiros e processador de ponto flutuante) é uma das razões, embora o processador não estivesse mais numa única pastilha. A ausência de microprogramação também é outra característica básica dessa arquitetura RISC que aumenta o seu desempenho.

6) Possibilita a simplificação do hardware que interpreta e executa as instruções. Assim, consegue-se diminuir o tempo necessário para a tarefa. Pode-se executar, assim, uma ou mais instruções por ciclo de relógio da UCP.

7) A arquitetura consiste em uma unidade de inteiros, uma de ponto flutuante, um co-processador, um gerenciador de memória e de uma memória cache. Essa fragmentação permite que os projetistas utilizem variações na construção de SC baseadas no processador. São cerca de 50 instruções (C. Op. de 6 bits), podendo haver até 512 registradores na UCP para utilização.

Índice

A

Ábaco, 9
Acerto (hit), 144
Adaptador, 377
AGP, 215
Álgebra Booleana, 87
Algoritmo, 5, 24
Análise léxica, 355
AND, 264
Armazenamento, unidade de, 122
Arquitetura, 2, 175
    RISC, 222
    superescalar, 234
    x86, 226
Arquivo, 31
ASCII, 263, 382
Assembly, 199

B

Barramento, 126, 372
    assíncrono, 217
    ciclo do, 215
    de controle, 212, 126
    de dados, 126, 181, 234
    de endereços, 126
    de expansão, 213, 373
    do sistema, 213, 373
    largura de um, 214
    local, 213, 373
    síncrono, 219
Base, 40, 442
    conversão de, 445
BCD (Binary Coded Decimal), 48, 307-308
Big endian, 254
Binária, 49
    divisão, 52
    multiplicação, 51
    soma, 49
    subtração, 49
BIOS (Basic Input Output System), 386
Bit, 29, 110
BIU (Bus Interface Unit), 232
Bloco, 400
BSC, 382
Byte, 29

C

Cache
    de RAM, 146
    L1, 146
    L2, 147
Caractere, 29, 262
CD-ROM, 424
Célula, 110
CENTRONICS, 383
Chip, 97
Chipsets, 246
Circuito integrado - CI, 97
Circuito somador, 204
Circuitos combinatórios, 93
CISC - Complex Instruction Set Computers, 175, 188, 429
Código, 29
    de operação, 189, 323, 352
    de representação, 460
    ligação, 258
    objeto, 258
    representação de caracteres, 29
Código-fonte, 359
Código-objeto, 351
Compilação, 353, 360
Compilador, 261, 274, 431
Complemento, 282, 285
    à base, 282
    à base menos um, 293-294
    a 2, 287
Computador digital, 64

D

Dados, 258, 261
Decodificador, 100
DIP (Dual Inline Package), 248
Disco(s)
    flexível, 405
    magnético, 402
    óticos, 424
    rígido, 402
    tempo de latência, 403
    tempo de SEEK, 403
    tempo de transferência, 403
Display, 387
Disquete
    fator de bloco, 406
    floppy-disk, 405
DMA - Acesso Direto à Memória, 413
DMA, 216

E

EBCDIC, 263
Endereçamento, 190
    modo de, 190, 323, 330, 334
        base mais deslocamento, 343
        direto, 332
        imediato, 331
        indexado, 338
        indireto, 333
    por registrador, 335
Endereço, 111, 122
ENIAC, 12, 13
Entrada, 26
Entrada/saída (E/S), 373-374, 384
    Acesso Direto à Memória (DMA - Direct Memory Access), 408
    interrupção, 408
    métodos, 408
    por programa, 408
Escrita (write), 125
Expressão lógica, 80

F

Falta (miss), 144
Fita magnética, 399
Flip-flop, 102, 156
FPU (Floating Point Unit), 203
Freqüência
    horizontal, 389
    vertical, 389

G

Gap semântico, 429
Gravação (write), 109

H

HDLC/SDLC, 382

I

IAS, 14
IEEE-754, 315
IESA, 215
Impressora(s)
    a laser, 395, 397
    coloridas, 421
    de cera aquecida, 395
    de impacto, 395
    de jato de tinta, 395, 397
    matriciais, 395
    por sublimação de tinta, 395, 423
    sem impacto, 395
    transferência térmica de cera e, 432
Instrução(ões), 26, 116
    assembly, 200
    ciclo, 170, 183, 191
        ADD Op, 197
    com dois operandos, 327
    com três operandos, 326
    com um operando, 329
    conjunto de, 322
    contador, 186
    de desvio, 365
    de máquina, 26, 170, 187, 323
    de um operando, 351
    decodificador, 186
    registrador, 186
    tamanho das, 325
Interface, 377

Interpretação, 359, 361
Interrupção, tratamento de, 413
ISA, 215

L

LCD, 420
Leitura (read), 109, 125
Leitura, ciclo de, 216
Ligação ou linkedição, 357-358
Linguagem(ens), 5
  ADA, 262
  de alto nível, 8, 349
  de máquina, 5, 347
  de montagem (Assembly Language), 348, 351
  de programação, 5, 347
Little-endian, 179, 254
Localidade, 143
  espacial, 143
  temporal, 143

M

Máquina de diferenças, 10
Memória(s), 26, 108, 113-115, 158
  Burst Extended Data Out (BEDO) DRAM, 158
  cache, 117, 144
  capacidade, 131
  ciclo, 114
  de acesso aleatório (ou randômico), 136
  Direct Rambus DRAM (DRDRAM), 158
  DRAM (Dynamic RAM), 137, 156
  EEPROM (Electrically ou Electronically EPROM), 140
  EPROM (Erasable PROM), 140
  Extended Data Out (EDO) DRAM, 158
  Fast Page Mode (FPM) DRAM, 158
  Flash ou Flash-ROM, 140
  meio magnético, 115
  meio ótico, 115
  principal, 119, 122
  PROM (Programmable Read Only Memory), 140
  RAM, 136, 158
  ROM (Read Only Memory), 137, 138
  secundária, 120
  SRAM (Static RAM), 137, 155
  Synchronous DRAM (SDRAM), 158
  volátil, 114
MFLOPS (milhões de operações de ponto flutuante por segundo), 18, 36, 433
Microcomputadores, 20
Microoperações, 222
Microprograma, 219
MIMD (Multiple Instruction stream, Multiple Data stream), 3
MIPS (milhões de instruções por segundo), 18, 432
MISD (Multiple Instruction stream, Single Data stream), 3
Montador, 201
Mouse, 407
  track-ball, 408

N

Notação
  científica, 297
  posicional, 39
Números fracionários, 272

O

8514A, 394
Operadores lógicos, 264, 266, 269
Operando, 323
Organização de um computador, 2
Overclocking, 251
Overflow, 207, 260, 273, 281, 296

P

Palavra, 30, 122, 177
  de dados, 377
Pastilha, 97
PC, 177
PCI, 215
Pentium, 235
Periféricos, 372
PGA (Pin Grid Array), 248
Pipeline, 172, 208
Pipelining, 181, 431, 434
Pixel, 392
Ponto flutuante, 300, 303
Porta(s), 67, 96
  AND, 67
  lógica (gate), 65
  NAND, 72
  NOR, 75
  NOT, 71
  OR, 69
  Wired-And, 96
  Wired-Or, 96
  XOR, 78
POWER (Performance Optimization With Enhanced RISC), 436
Power PC, 428
Processador central, 168
Processamento
  de dados, 1, 172
  eletrônico de dados, 1
Programa, 4, 31, 347
  de computador, 25

R

RAM (Random Access Memory), 118, 123
Recarregamento (refresh), 157
Registrador(es), 116, 126, 175
  de dados, 175
  de Dados da Memória (RDM), 126, 187
  de Endereços da Memória (REM), 126, 177
  instruções, 116
  PSW (Program Status Word), 177
Registro, 31
  físico, 400
Relógio, 182, 183
  freqüência, 184
Representação, 274
  de dados, 260
  decimal, 307
  normalizada, 299
  ponto fixo, 274
  ponto flutuante, 297
RISC - Reduced Instruction Set Computers, 172, 175, 188, 428
ROM (Read Only Memory), 124
RS/6000, 428, 436

S

Scanners, 423
SEAC (Standard Eastern Automatic Computer), 324
SEC (Single Edge Contact), 249
SIMD (Single Instruction stream, Multiple Data stream), 3, 172
Sinal e magnitude, 275-276
SISD (Single Instruction stream, Single Data stream), 3, 172, 184, 208
Sistema, 3
  de computação, 371
  de numeração, 440
  posicional, 440
  processamento de dados, 4
SPARC, 428, 435
SPECmark, 433
SRAM, 118
SVGA, 394

T

Tabela
  de símbolos, 353
  verdade, 65
Teclado, 384, 415
  debouncing, 416
  QWERTY, 417
Tempo de acesso, 108, 113, 116, 127
Temporariedade, 115
TFT, 420
Tipo numérico, 272
Transmissão
  assíncrona, 380
  paralela, 376, 383
  serial, 376, 379
  síncrona, 382

U

UAL, 174, 203
UC, 219
UNICODE, 263
Unidade
  Aritmética Lógica - UAL (ALU - Arithmetic and Logic Unit), 116
  Central de Processamento - UCP, 26
  de Controle (UC), 182
USART, 383
USB, 215, 383

V

Vazão, 37
VGA, 394
Vídeo, 386, 418
  colorido, 393
  de gás plasma, 421
  dot pitch, 394
  modo entrelaçado, 393
  não-entrelaçado, 393
  placa de, 390
  resolução de, 394
Volatilidade, 114
VRC, 387